

Développement de Jeux Vidéo (en Java)

Grandement inspiré de
« Killer game programming in java »

<http://fivedots.coe.psu.ac.th/~ad/jg/>

vincent.thomas@loria.fr

MdC IUT Charlemagne - LORIA / équipe MAIA

- **Bibliothèques existantes**
 - Par exemple, Slick2D
- **Objectif présentation**
 - comprendre comment jeu fonctionne
 - Développer un jeu est relativement simple
- **Jeu 2D - Sans bibliothèque**
 - À partir des classes JAVA
 - Affichage sprite et animation avec Swing

- L'idée du siècle dernier

Introduction

- L'idée du siècle dernier



- L'idée du siècle dernier



Introduction



Nouveau JEU révolutionnaire !!! (en 1990)

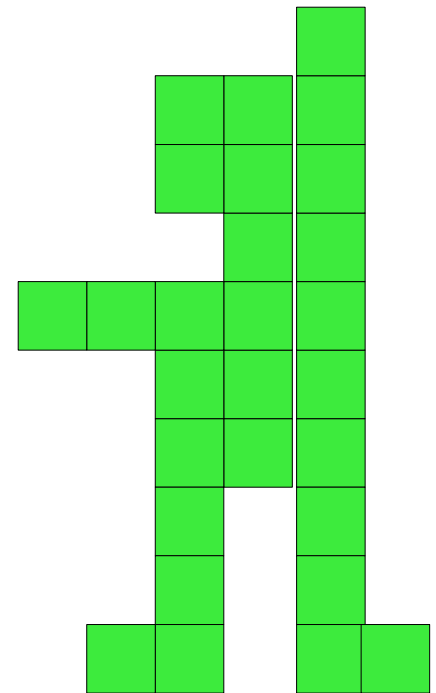
Introduction



Gestion de sprites



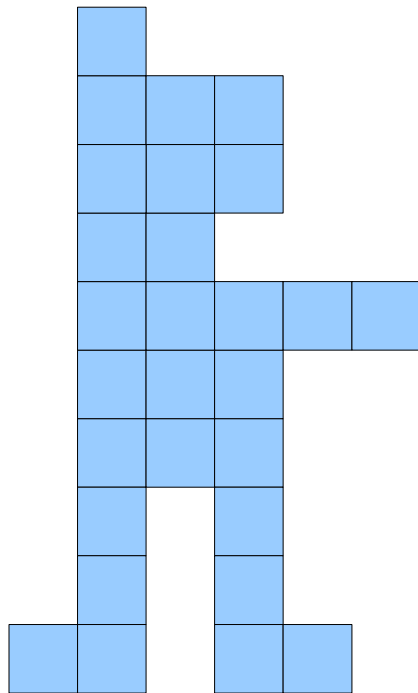
=





=

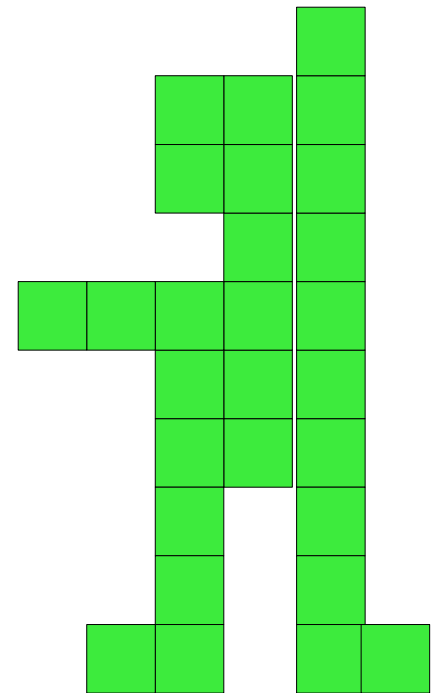
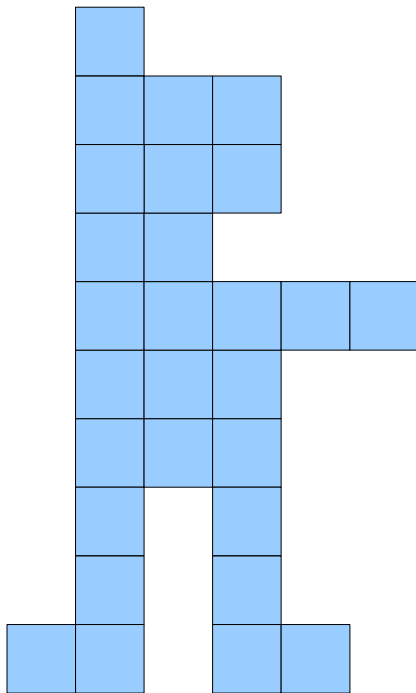
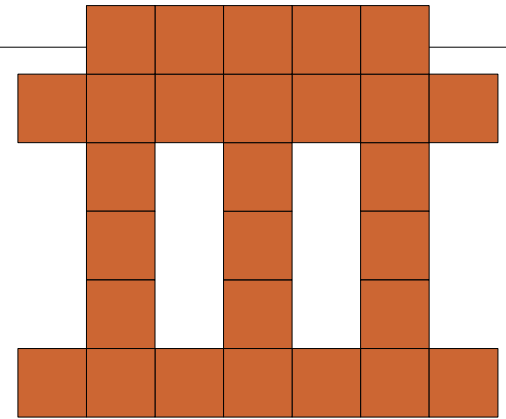


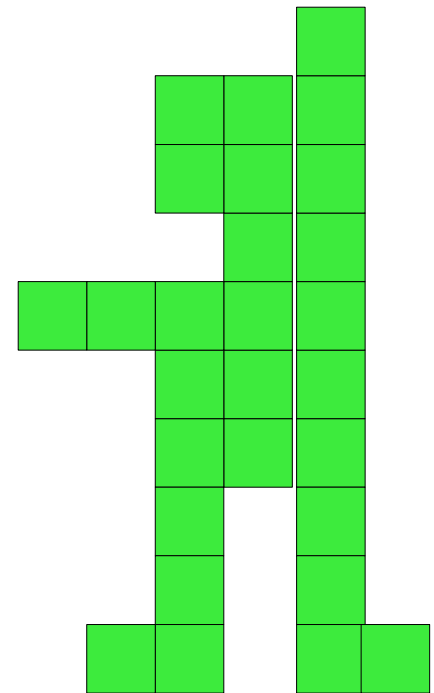
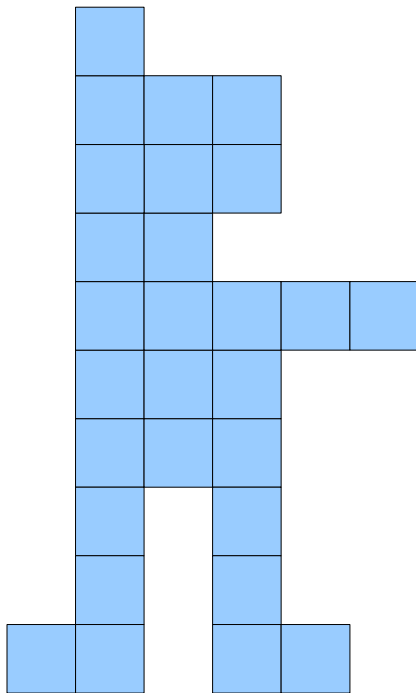


=



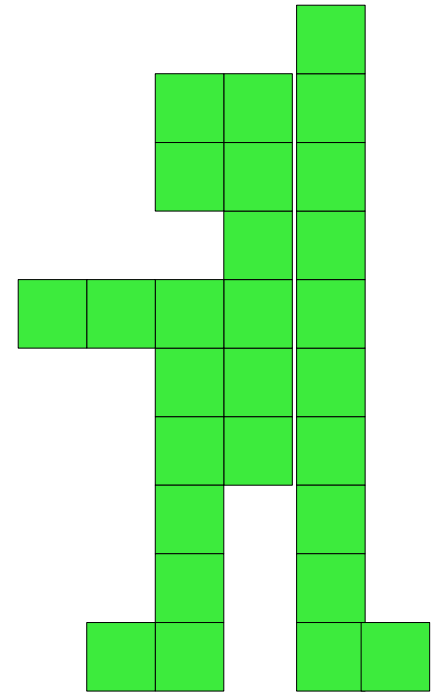
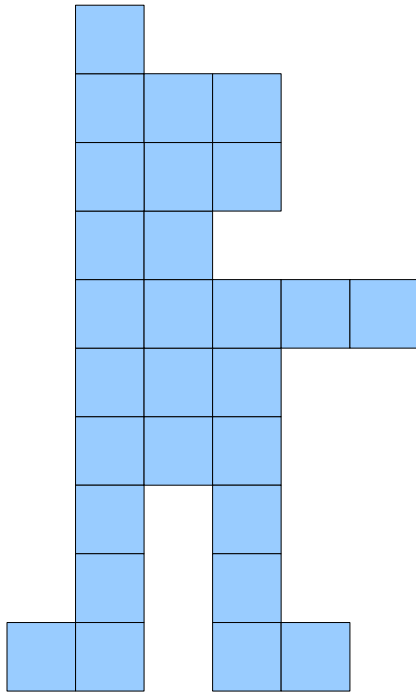
Gestion de décor



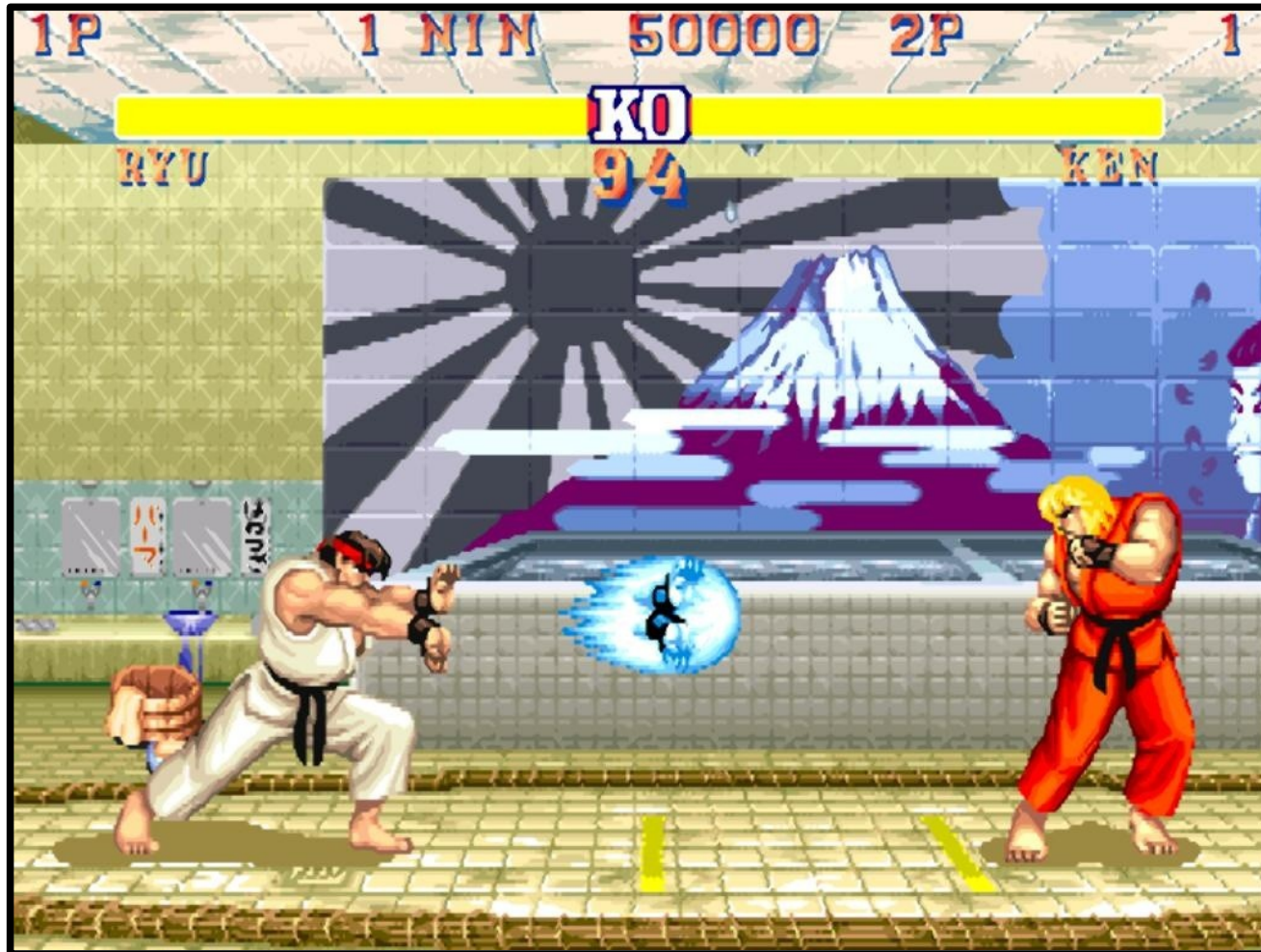


Gestion du contrôle et déplacement

```
10 : lecture manette  
20 : mise en attente  
30 : goto 10
```

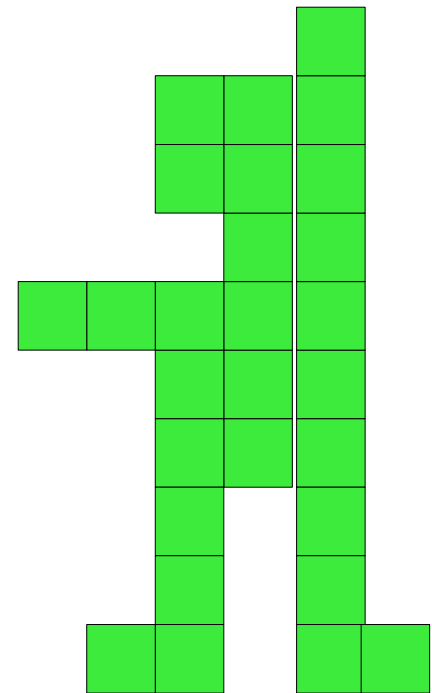
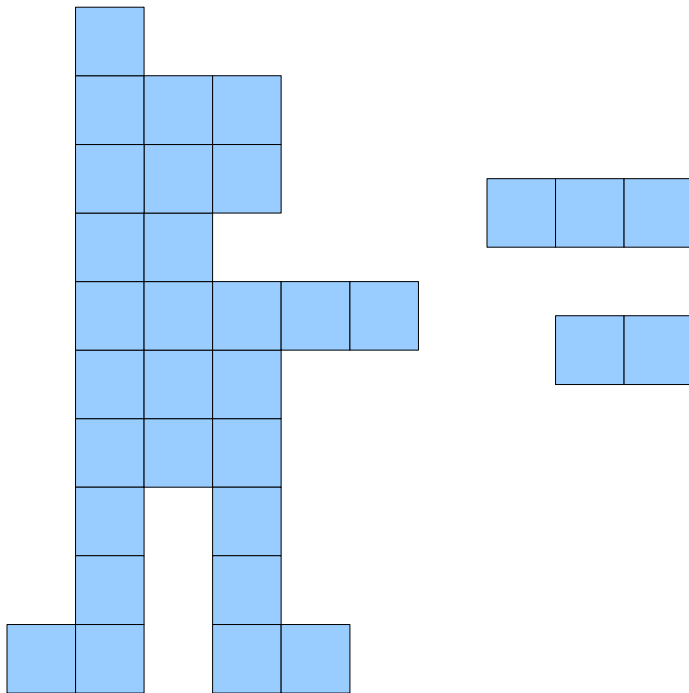


Introduction



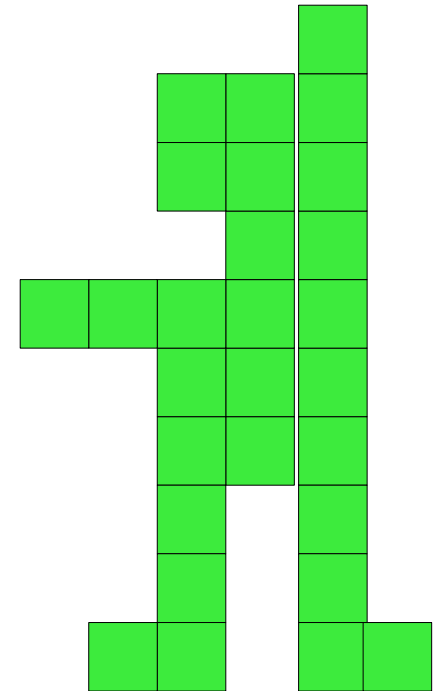
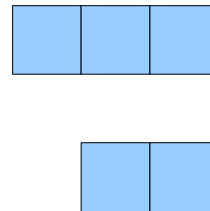
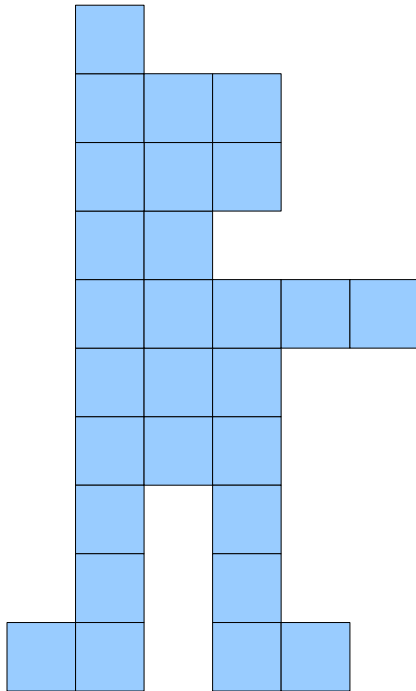
Gestion éléments dynamiques

```
10 : i = 0  
20 : i++  
30 : If (i<40) GOTO 10
```



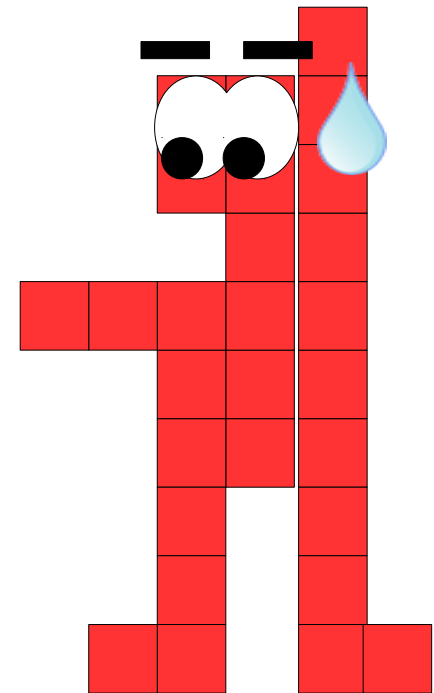
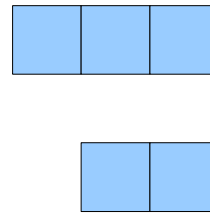
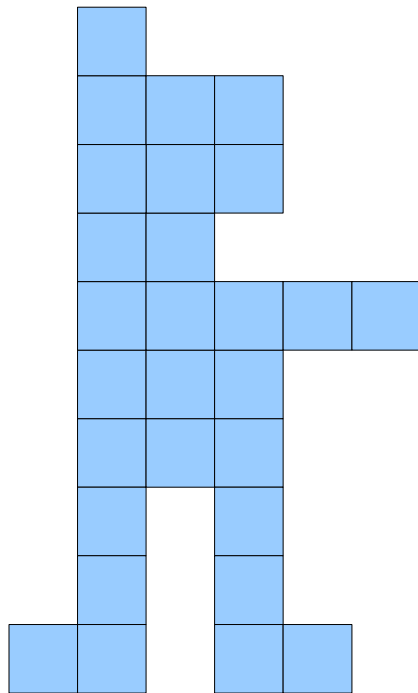
Gestion éléments dynamiques

```
10 : i = 0  
20 : i++  
30 : If (i<40) GOTO 10
```



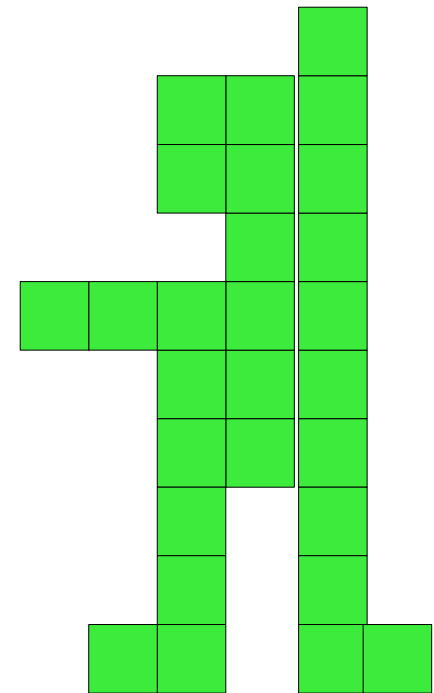
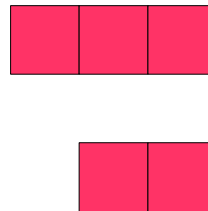
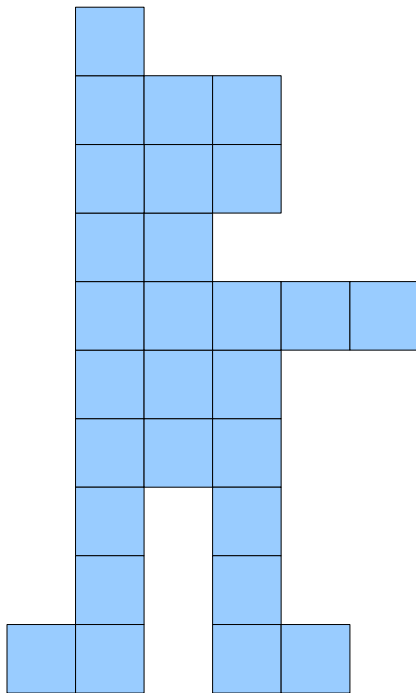
Problème lié au contrôleur

```
10 : i = 0  
20 : i++  
30 : If (i<40) GOTO 10
```



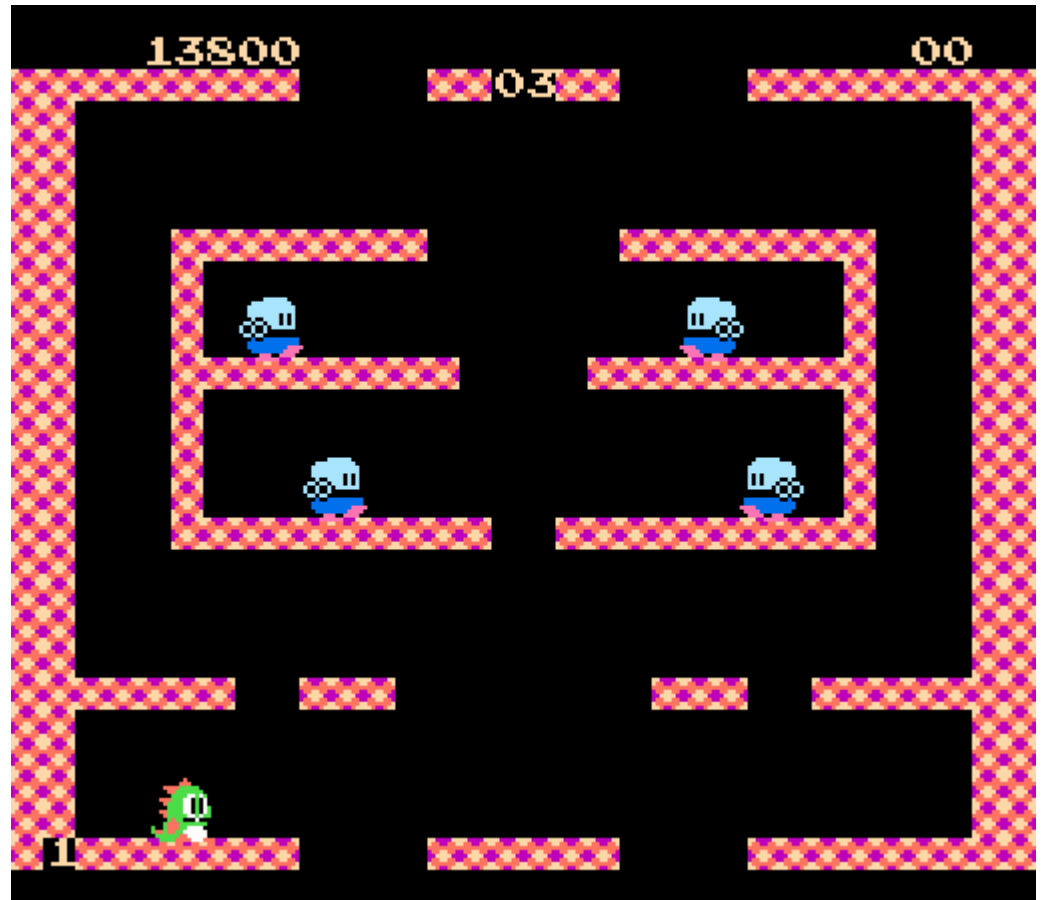
```
10 : i = 0  
20 : i++  
30 : If (i<40) GOTO 10
```

```
10 : lecture manette  
20 : mise en attente
```





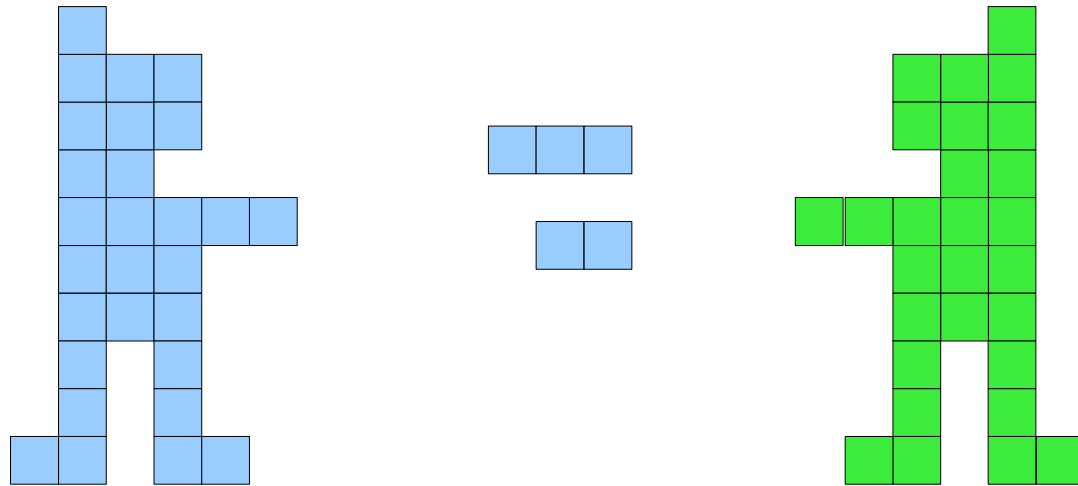
??????????



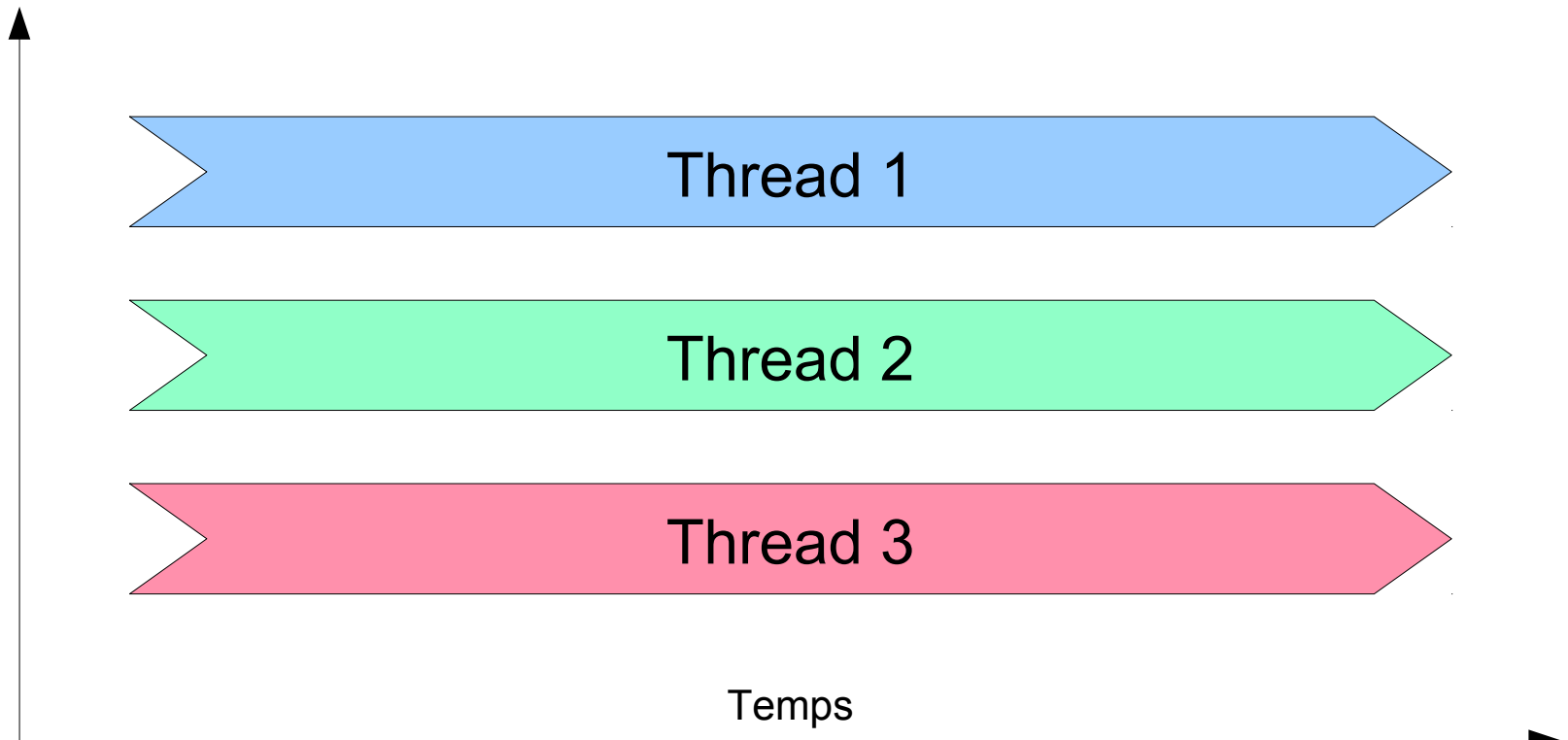
- Comment faire pour évoluer en parallèle ?

Comment gérer plusieurs personnages

- En même temps ?

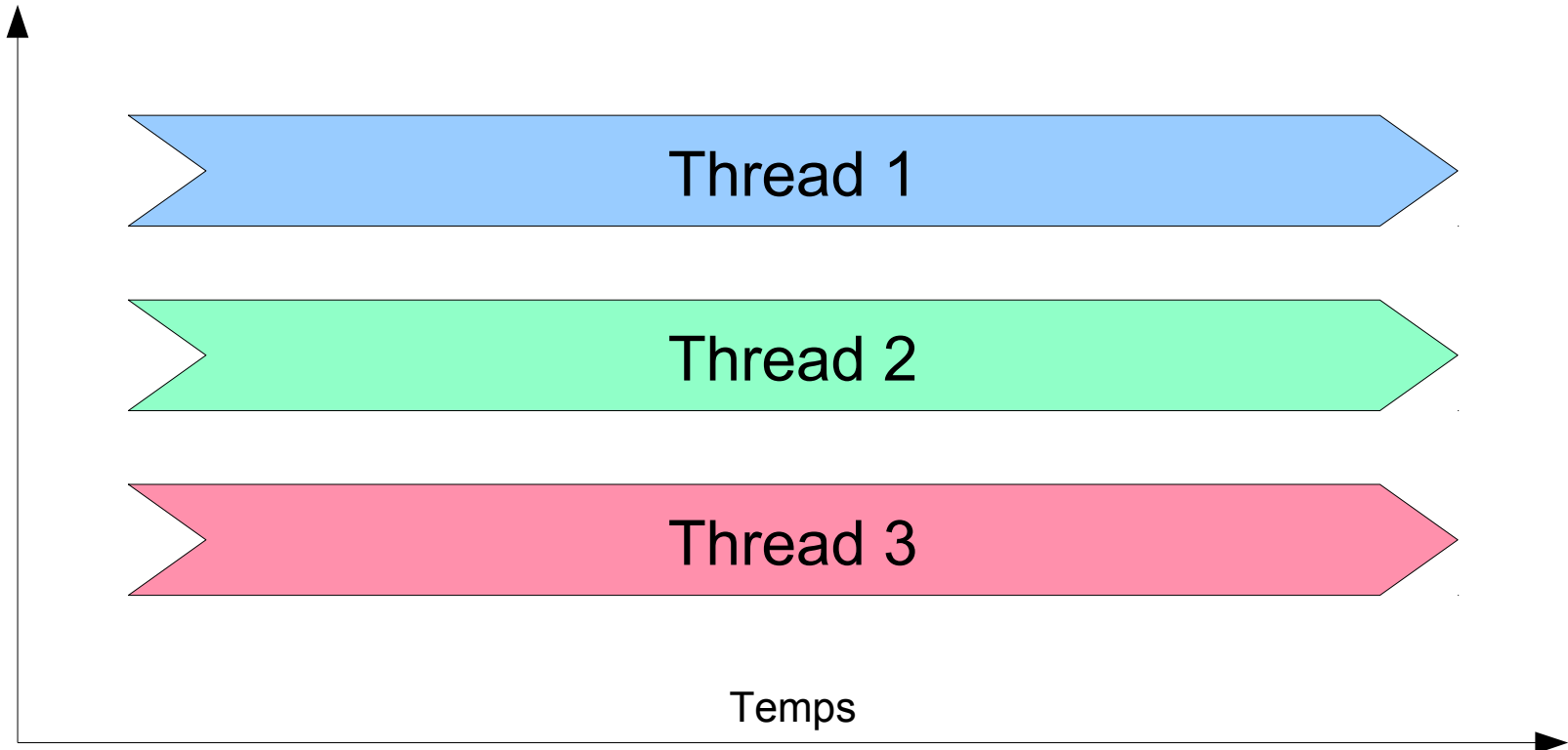
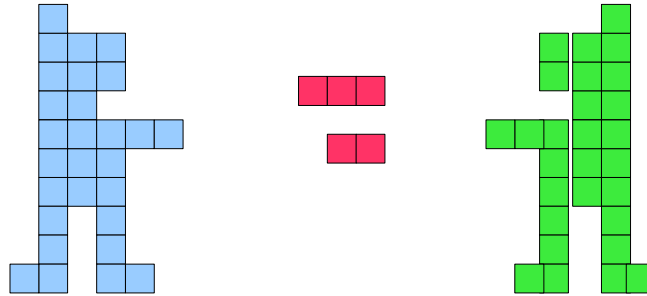


- Première (mauvaise) reponse : les Thread
 - Processus exécutés en parallèle
 - Des thread de base (ex JComponent)



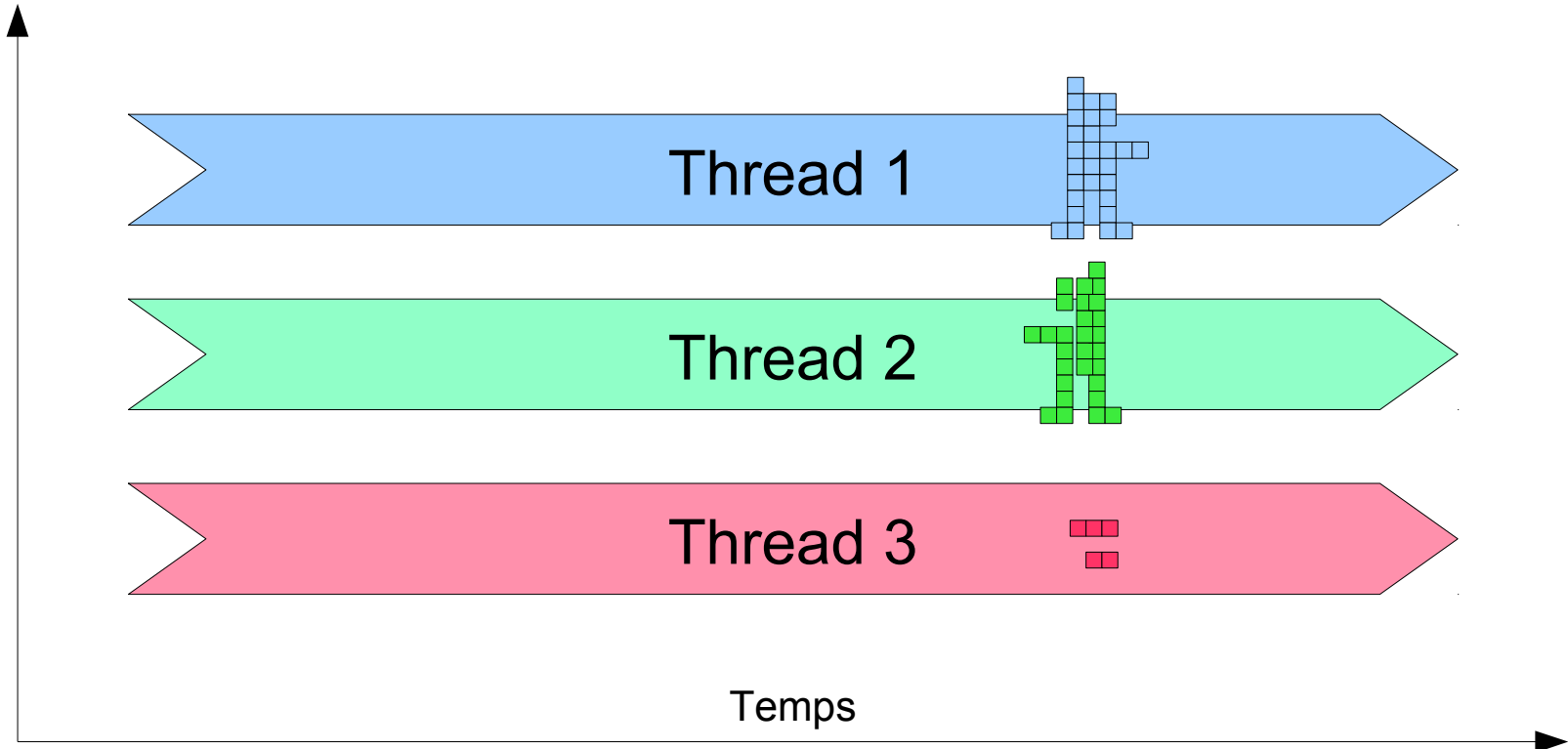
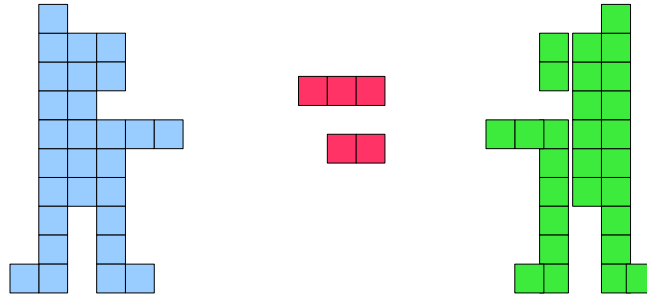
Thread

```
10 : i = 0  
20 : i++  
30 : If (i<40) GOTO 10
```



Thread

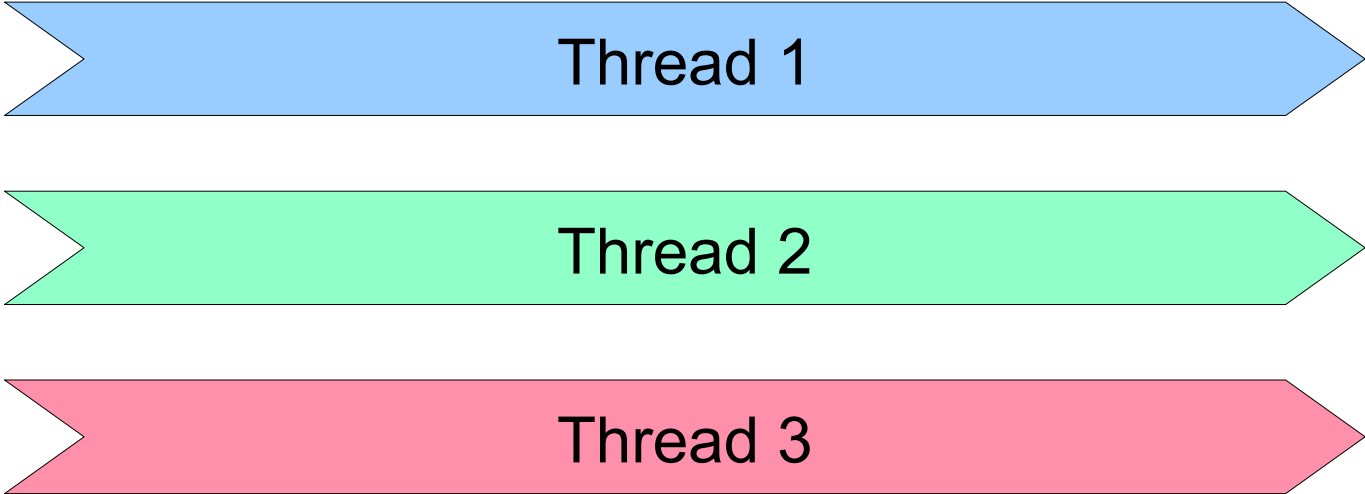
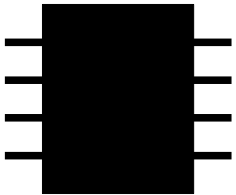
```
10 : i = 0  
20 : i++  
30 : If (i<40) GOTO 10
```



- Thread

- Processus exécutés en parallèle
- Des thread de base (ex JComponent)

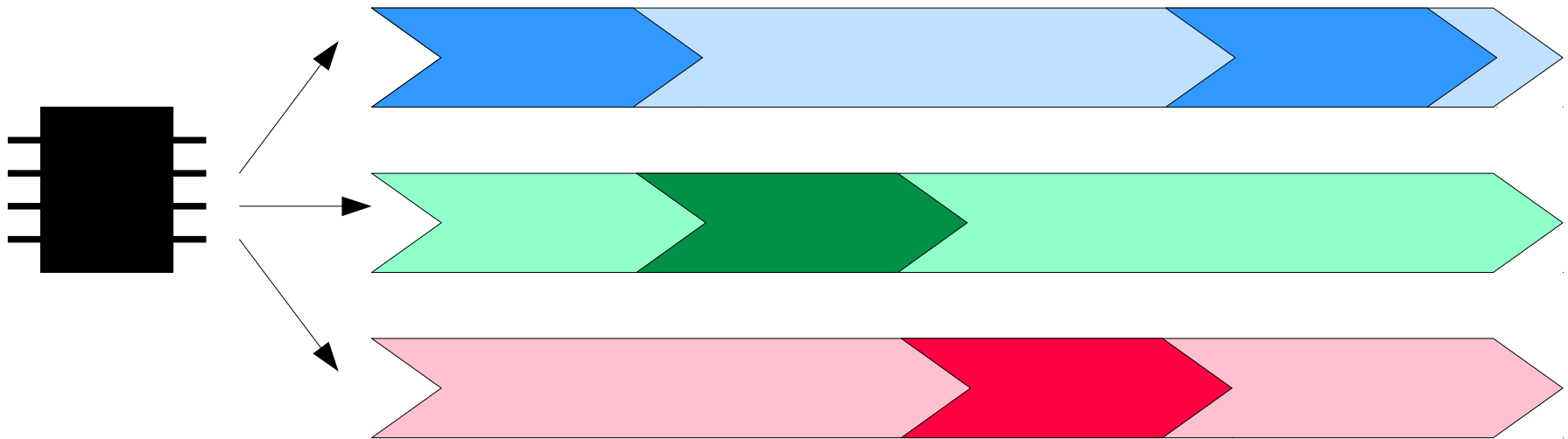
abstraction



- Thread

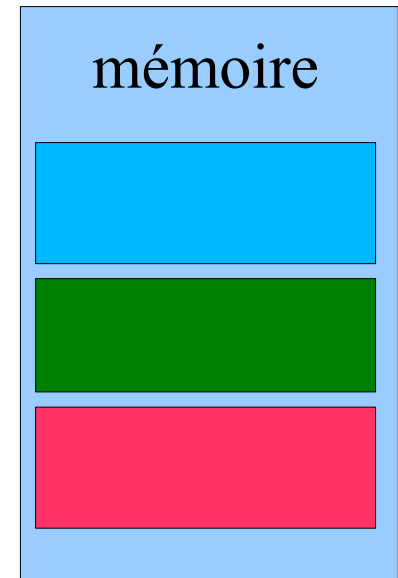
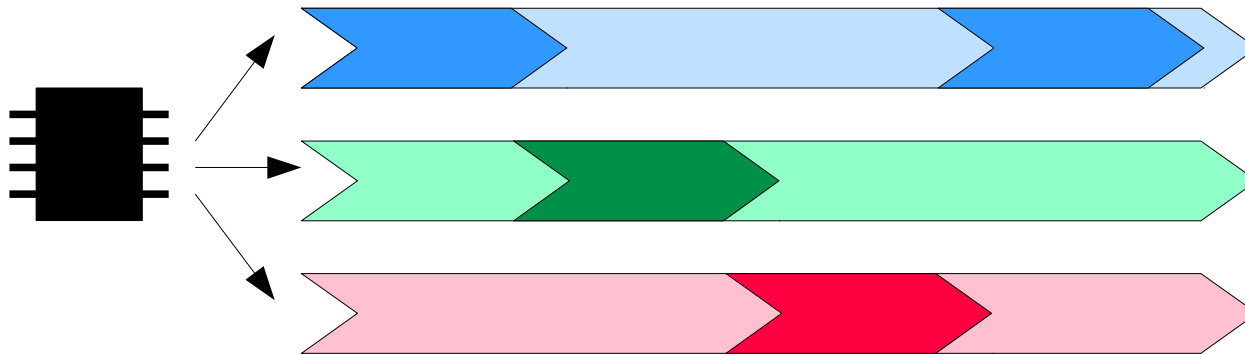
- Processus exécutés en parallèle
- Des thread de base (ex JComponent)

abstraction



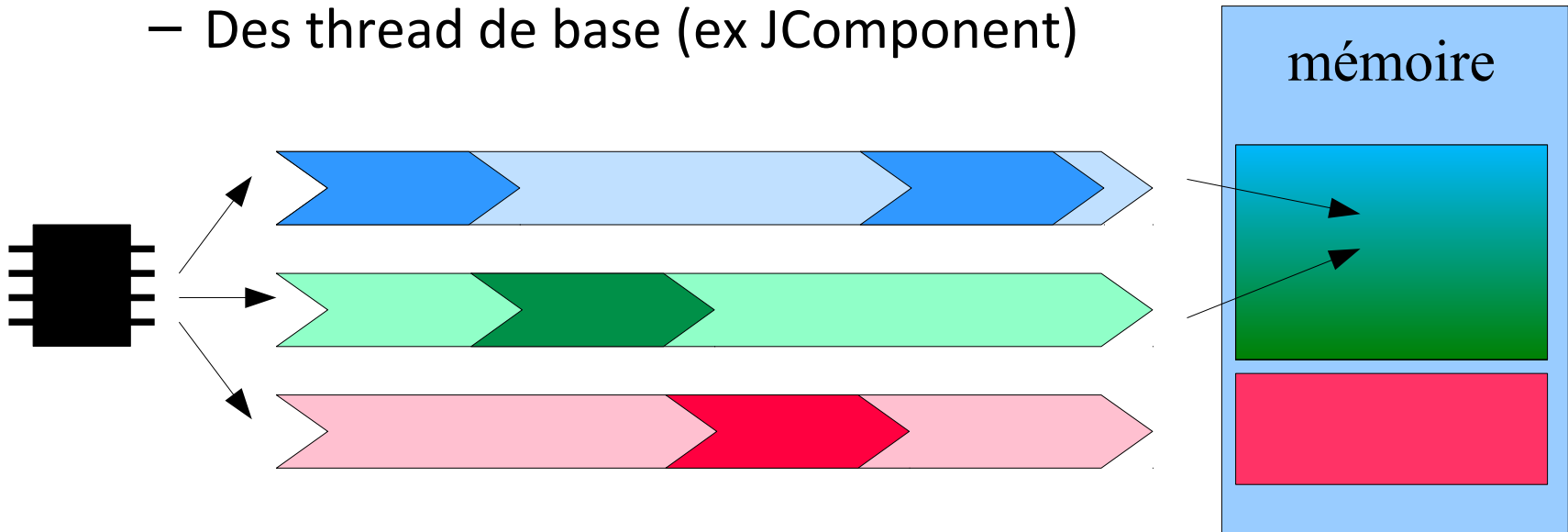
- Thread

- Processus exécutés en parallèle
- Des thread de base (ex JComponent)



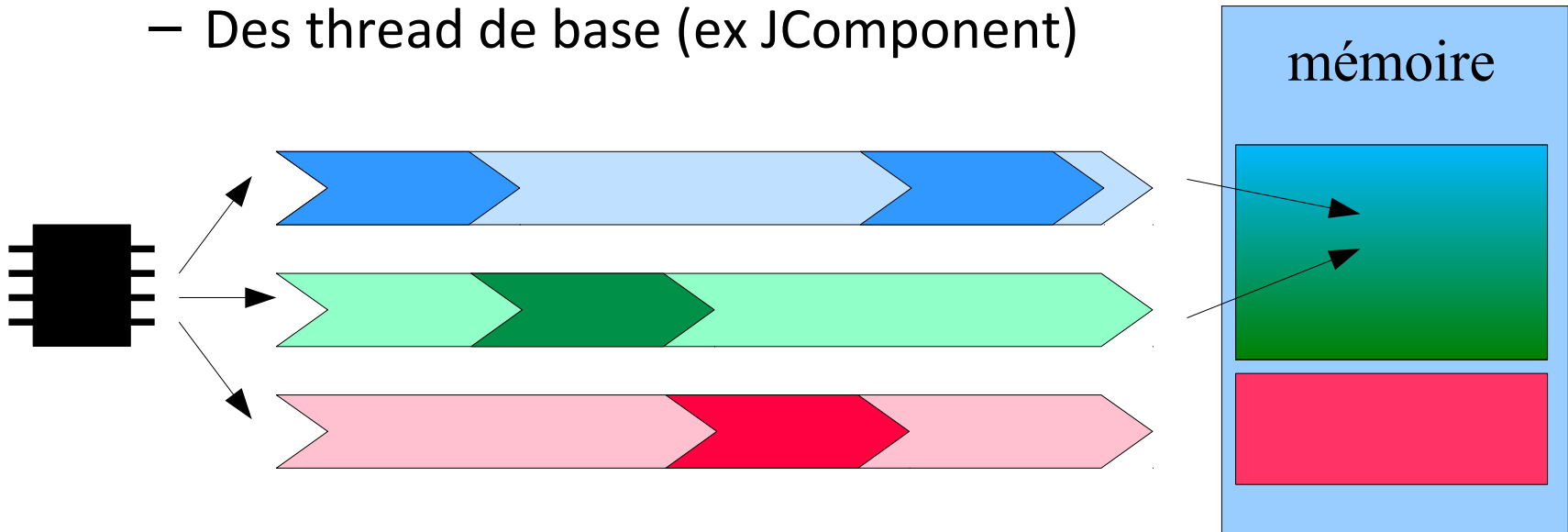
- Thread

- Processus exécutés en parallèle
- Des thread de base (ex JComponent)



- Thread

- Processus exécutés en parallèle
- Des thread de base (ex JComponent)

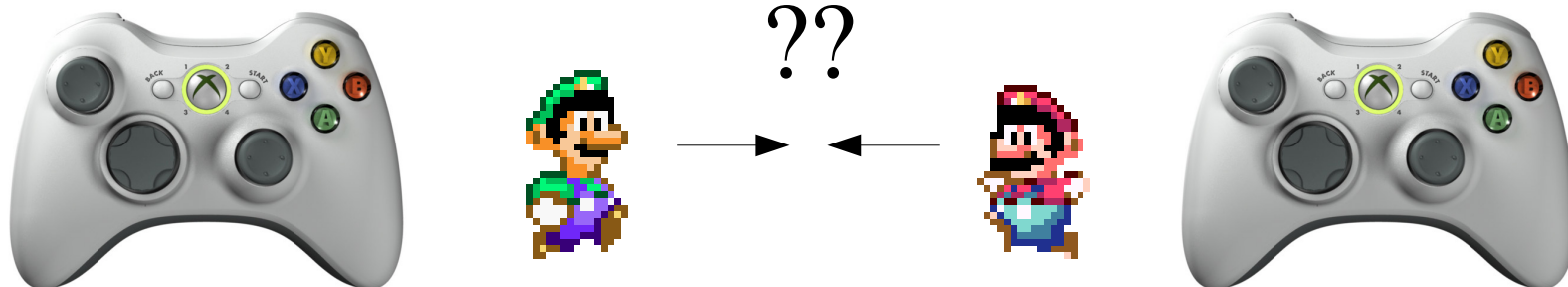


- Problèmes des threads

- Méthodes ré-entrantes, Concurrence d'accès
- Synchronisation, stop/pause

Thread

- **Thread**
 - Processus exécutés en parallèle
 - Des thread de base (ex JComponent)
- **Problèmes des threads**
 - Méthodes réentrantes, Concurrence d'accès
 - Synchronisation, stop/pause



Thread



??



```
If ( p [ Luigi.x + 1 ] == libre)
{
    Luigi.x=Luigi.x+1
    p [ Luigi.x ] = occupé
}
```

```
If ( p [ Mario.x - 1 ] == libre)
{
    Mario.x=Mario.x-1
    p [ Mario.x ] = occupé
}
```

Thread



??



```

If ( p [ Luigi.x + 1 ] == libre)
{
    Luigi.x=Luigi.x+1
    p [ Luigi.x ] = occupé
}

```

```

If ( p [ Mario.x - 1 ] == libre)
{
    Mario.x=Mario.x-1
    p [ Mario.x ] = occupé
}

```

```

If ( p [ Luigi.x + 1 ] == libre)
{
    Luigi.x=Luigi.x+1
    ←
    p [ Luigi.x ] = occupé
}

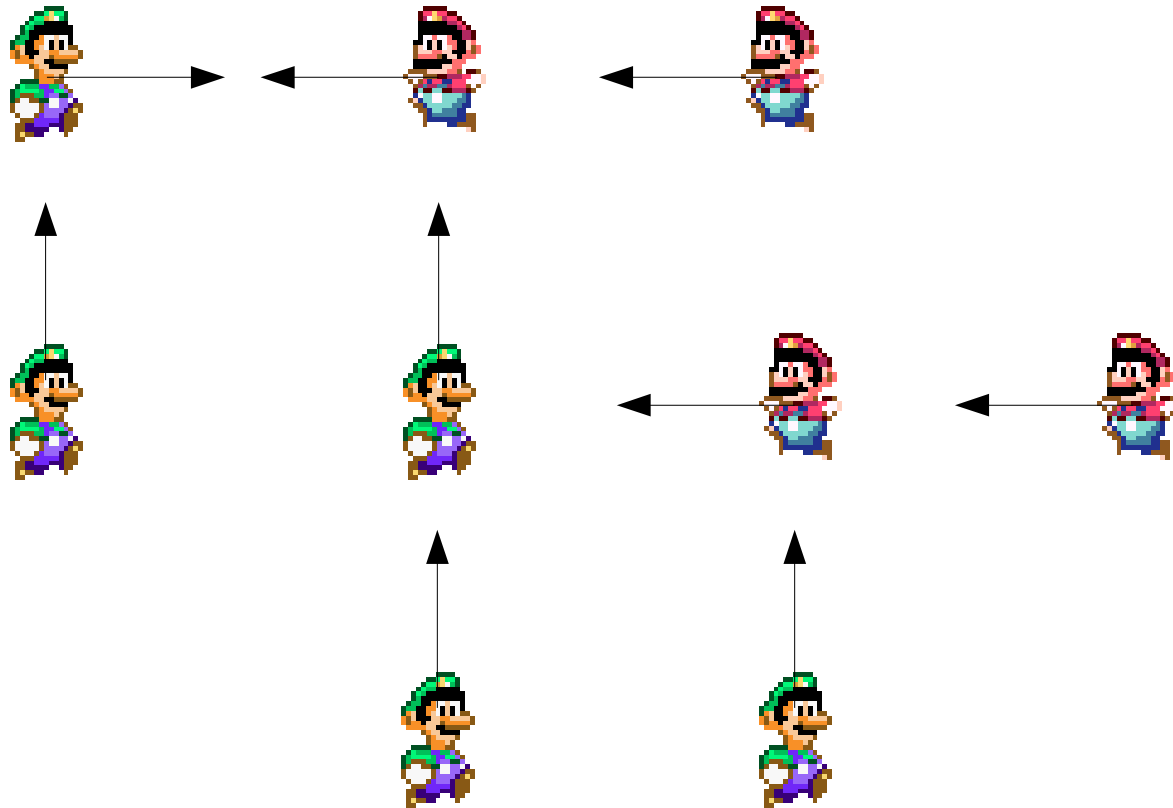
```

```

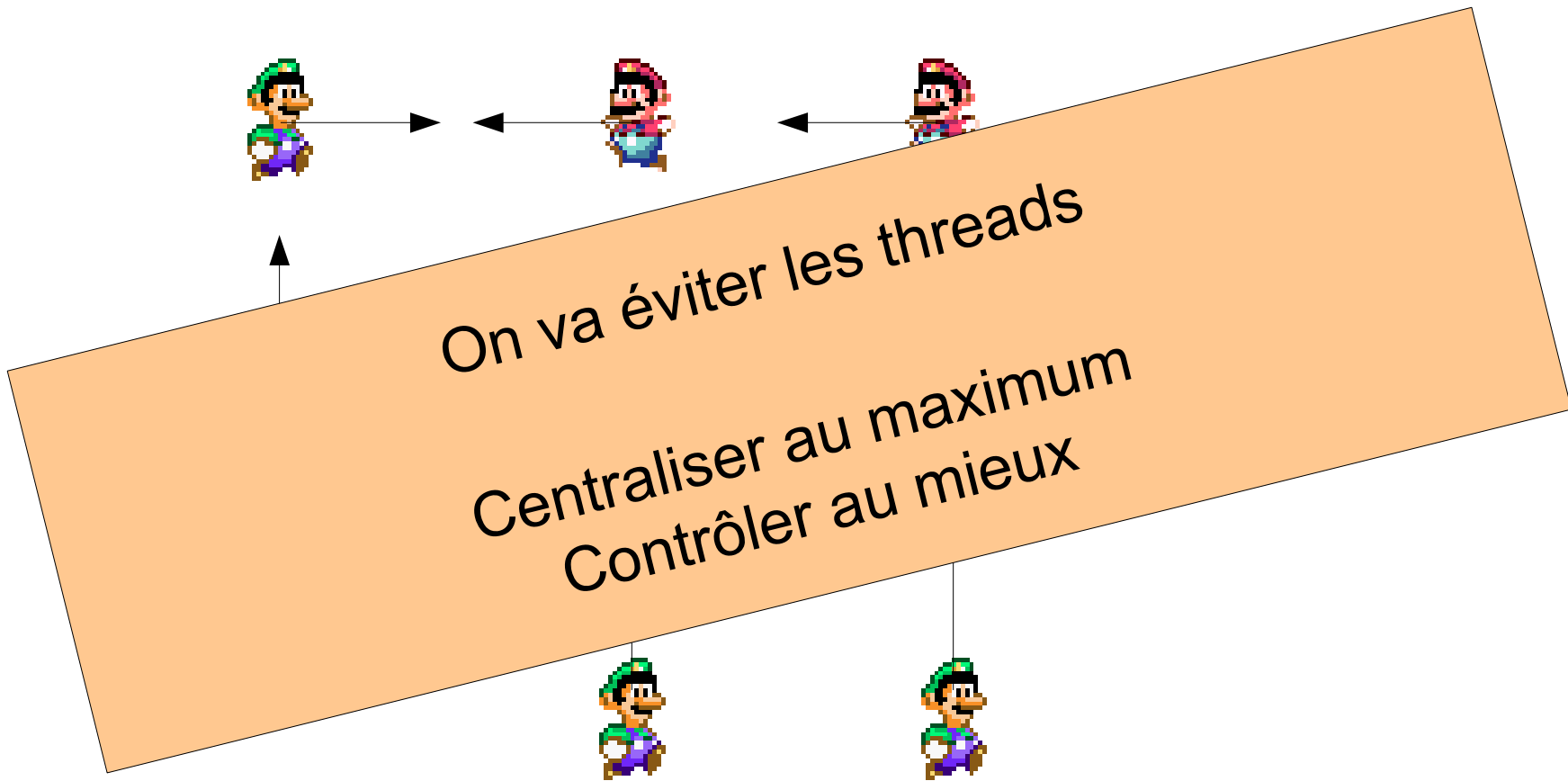
If ( p [ Mario.x - 1 ] == libre)
{
    Mario.x=Mario.x-1
    p [ Mario.x ] = occupé
}

```


- Problèmes des threads
 - amplifié quand beaucoup de personnages



- Problèmes des threads
 - amplifié quand beaucoup de personnages

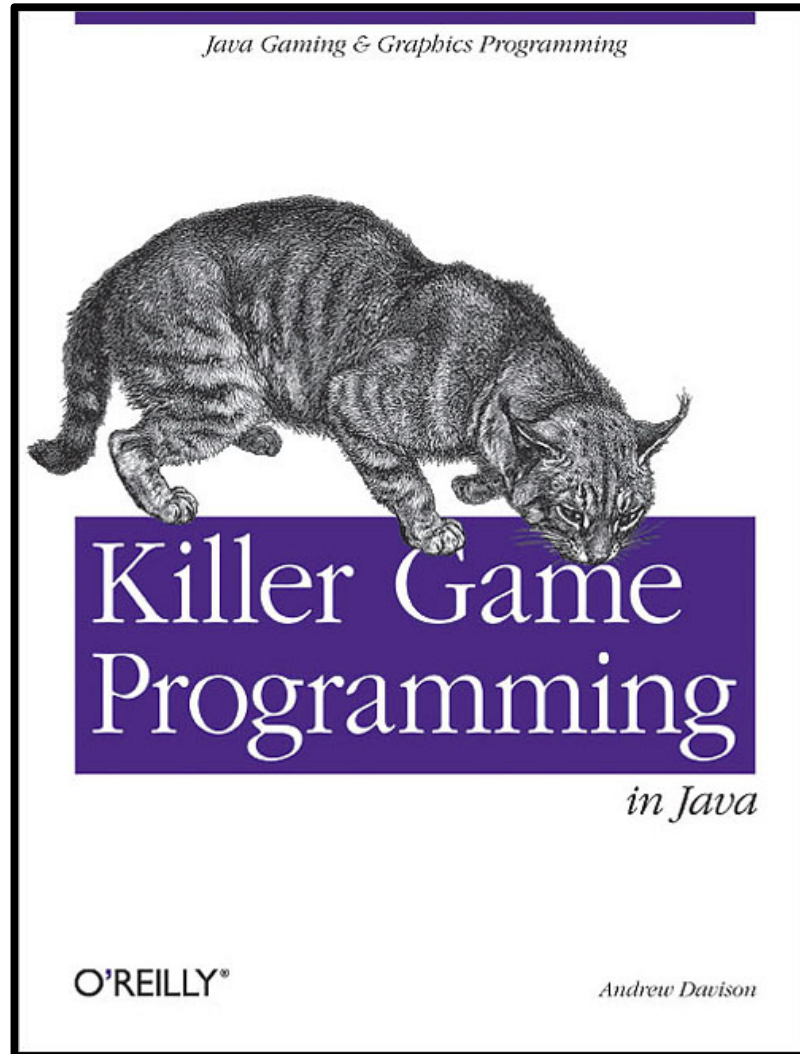


- **Utilité des thread**
 - Processus longs interruptibles
- **Exemples**
 - Musique
 - Intelligence artificielle
 -

- Comment faire pour évoluer en parallèle ?



Une (bonne) référence



<http://fivedots.coe.psu.ac.th/~ad/jg/ch1/index.html>

- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
- Réseau

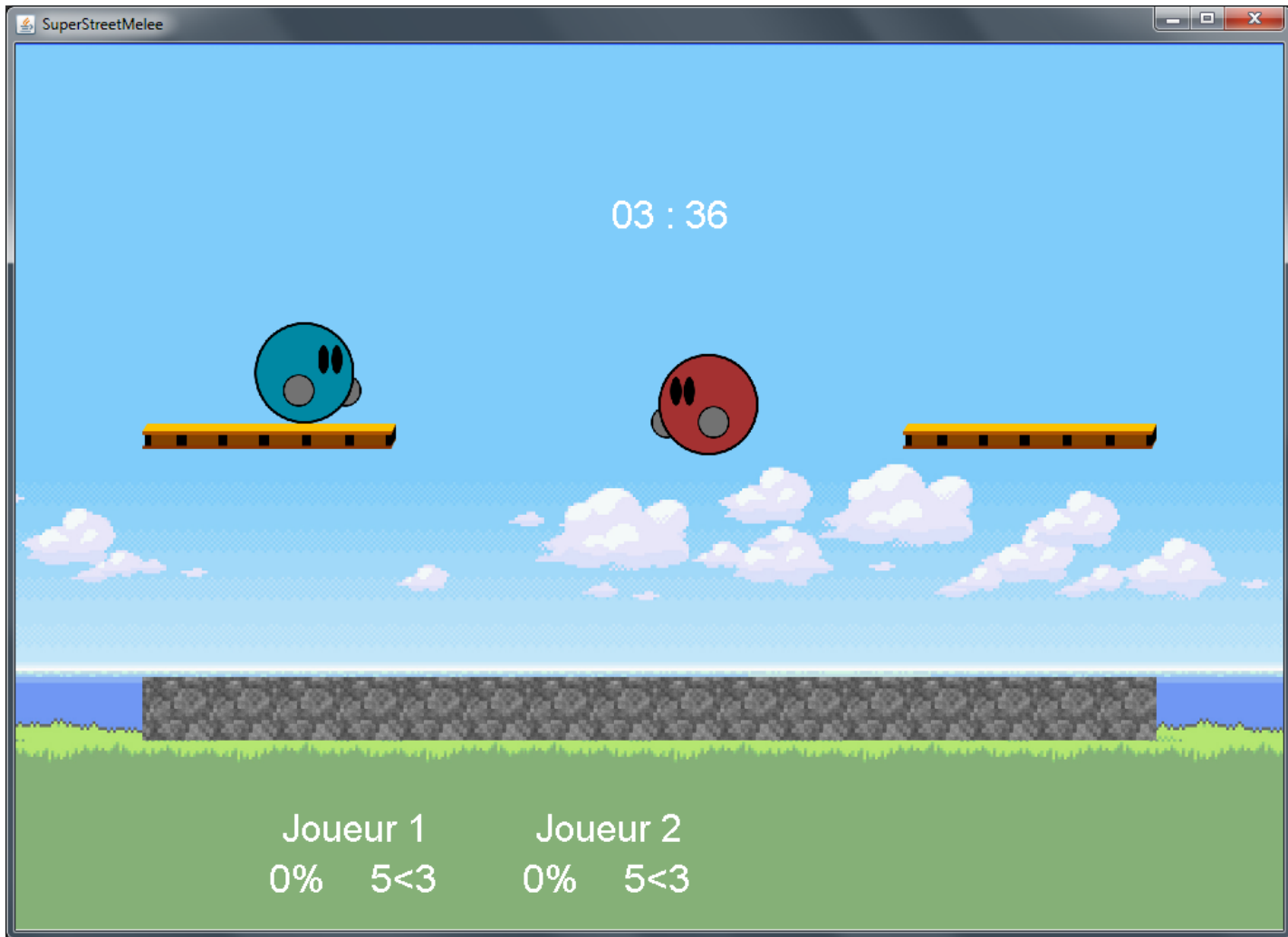
Application Fil Rouge





Espace et temps continus

Exemple de jeu – Super Street Melee

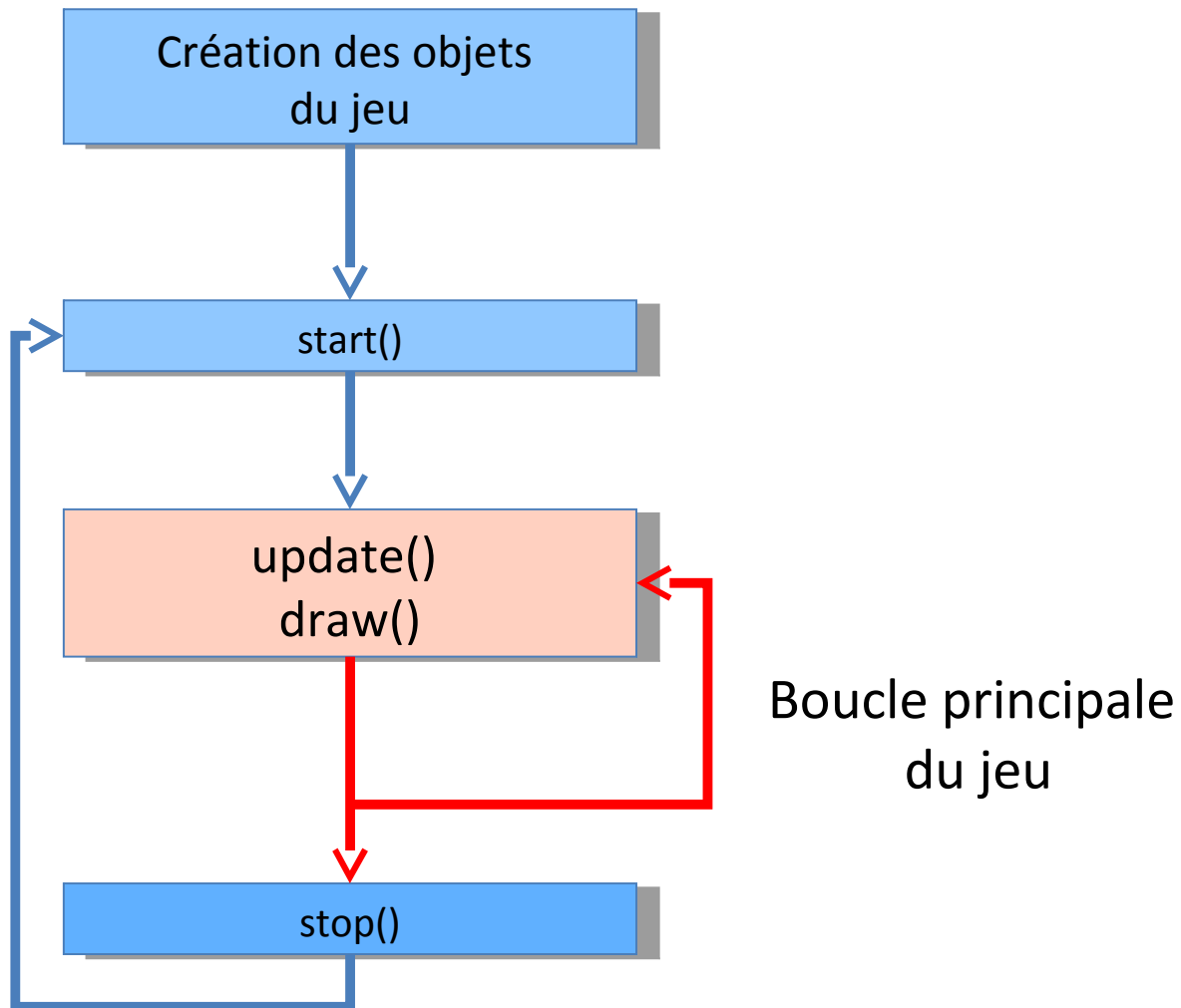


Projet DUT – 2eme année (2014-2015)

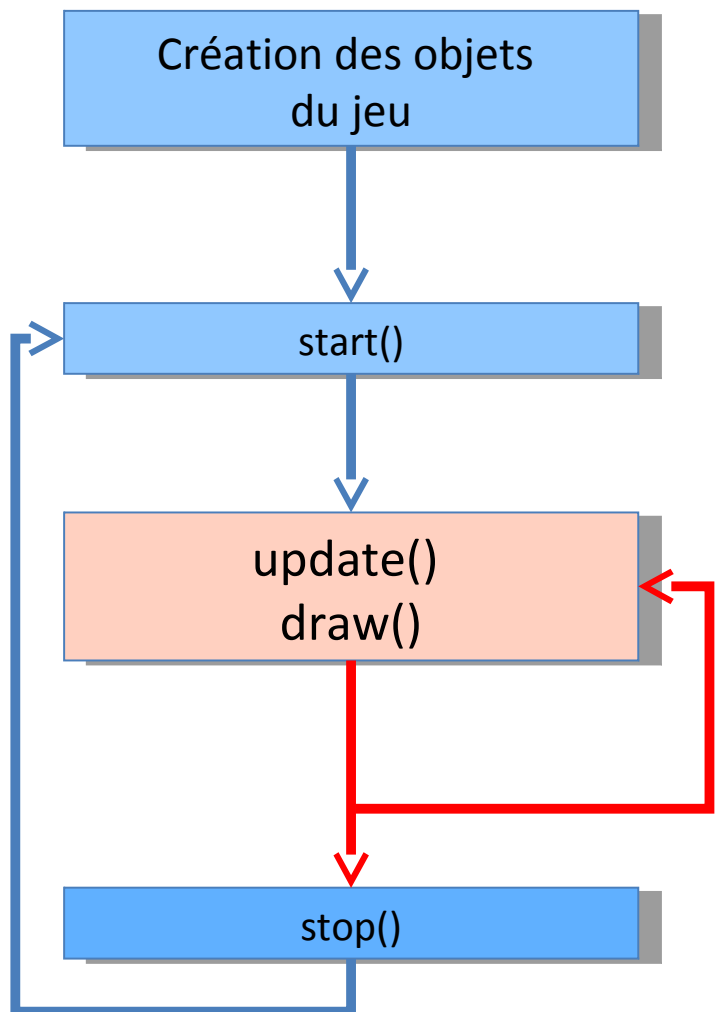
BESSION Léonard, CORNAT Jacques, LUC Aymeric et RAULOT Adrien

- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
- Réseau

Boucle de jeu



Boucle de jeu



FPS

Frames par seconde

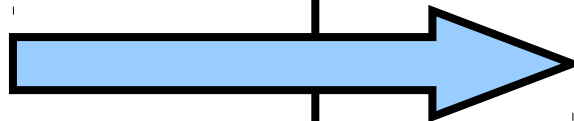
Objectif 60 fps

- **Classe Game: deux méthodes**
 - Update: mise à jour du jeu
 - Render: affichage du jeu

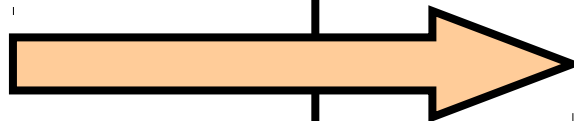
- **Classe Principale**
 - Boucle boolean fini (gagne/perdu)
 - Appels méthodes de Game
 - Méthode update
 - Méthode render

Exemple de jeu

```
public class Game {  
  
    int n = 0;  
    JPanel p;  
    boolean fini = false;  
  
    public void update() {  
        n = n + 1;  
        if (n > 10000)  
            fini = true;  
    }  
  
    public void render() {  
        p.repaint();  
    }  
}
```



Mise à jour



Affichage

Surcharge de paint

- Dessin d'un cercle
- Affichage de n

Exemple de jeu

Classe Principale

```
public class Game {  
  
    int n = 0;  
    JPanel p;  
    boolean fini = false;  
  
    public void update() {  
        n = n + 1;  
        if (n > 10000)  
            fini = true;  
    }  
  
    public void render() {  
        p.repaint();  
    }  
}
```

Classe Principale

```
public class Game {  
  
    int n = 0;  
    JPanel p;  
    boolean fini = false;  
  
    public void update() {  
        n = n + 1;  
        if (n > 10000)  
            fini = true;  
    }  
  
    public void render() {  
        p.repaint();  
    }  
}
```

```
public class Princ1 {  
  
    static Game g;  
    // prog principal  
    public static void main(String[] args) {  
        // creation du jeu  
        g = new Game();  
        // appel à la boucle  
        boucle();  
    }  
  
    // boucle de jeu  
    public static void boucle() {  
        while (g.fini == false) {  
            g.update();  
            g.render();  
        }  
    }  
}
```


Exemple de jeu

Classe Principale

```
public class Game {  
  
    int n = 0;  
    JPanel p;  
    boolean fini = false;  
  
    public void update() {  
        n = n + 1;  
        if (n > 10000)  
            fini = true;  
    }  
  
    public void render() {  
        p.repaint();  
    }  
}
```

```
public class Princ1 {  
  
    static Game g;  
    // prog principal  
    public static void main(String[] args) {  
        // creation du jeu  
        g = new Game();  
        // appel à la boucle  
        boucle();  
    }  
  
    // boucle de jeu  
    public static void boucle() {  
        while (g.fini == false) {  
            g.update();  
            g.render();  
        }  
    }  
}
```

Exemple de jeu

Classe Principale

```
public class Game {

    int n = 0;
    JPanel p;
    boolean fini = false;

    public void update() {
        n = n + 1;
        if (n > 10000)
            fini = true;
    }

    public void render() {
        p.repaint();
    }

}
```

```
public class Princ1 {

    static Game g;
    // prog principal
    public static void main(String[] args) {
        // creation du jeu
        g = new Game();
        // appel à la boucle
        boucle();
    }

    // boucle de jeu
    public static void boucle() {
        while (g.fini == false) {
            g.update();
            g.render();
        }
    }

}
```

Moteur générique

- Moteur générique
 - v01.02

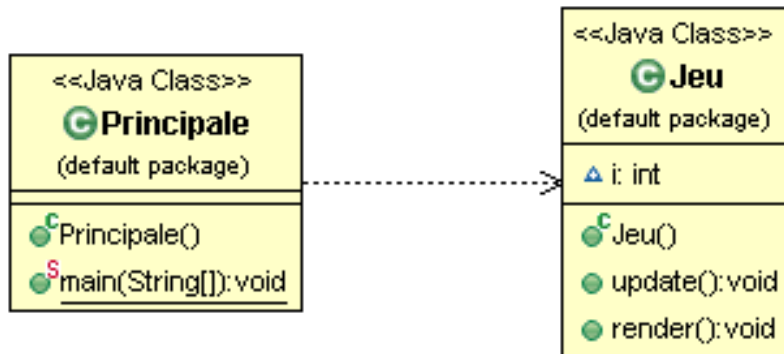
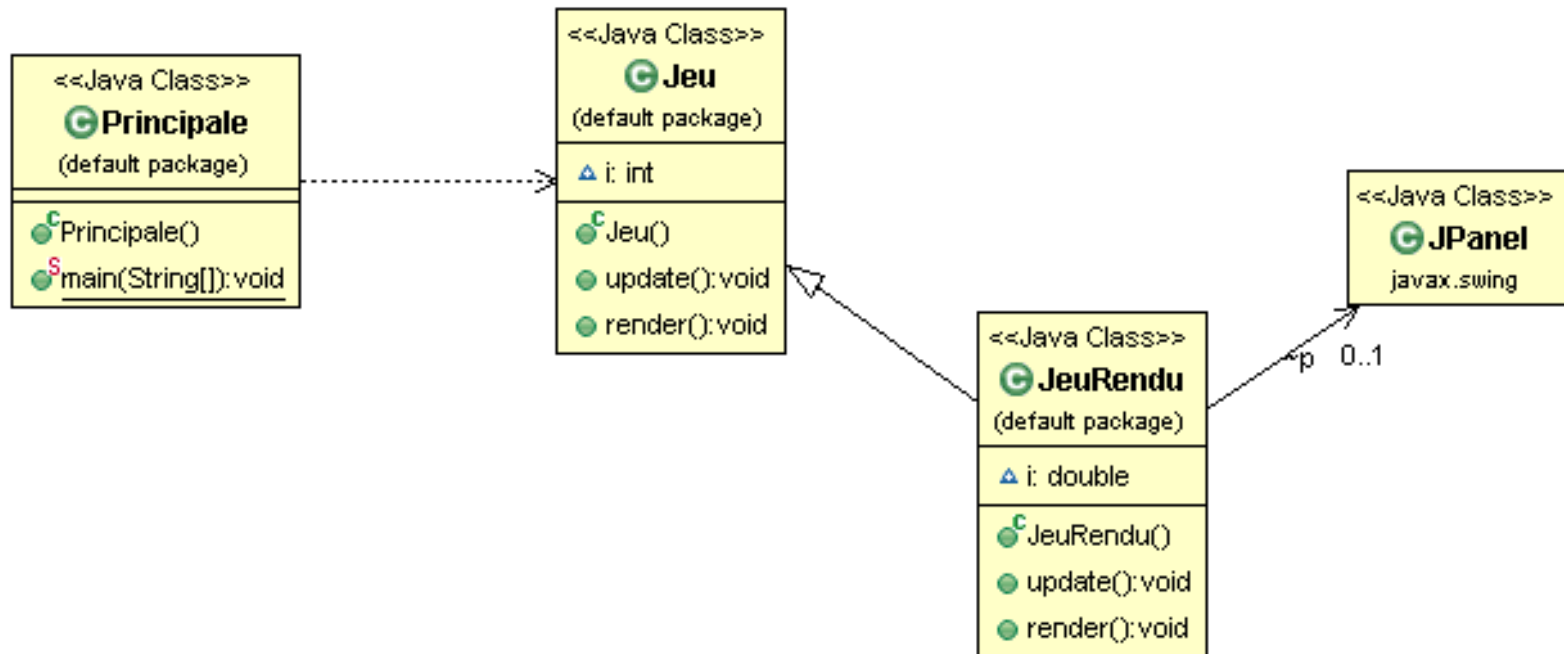


Diagramme de classe

- Moteur générique
 - v01.03

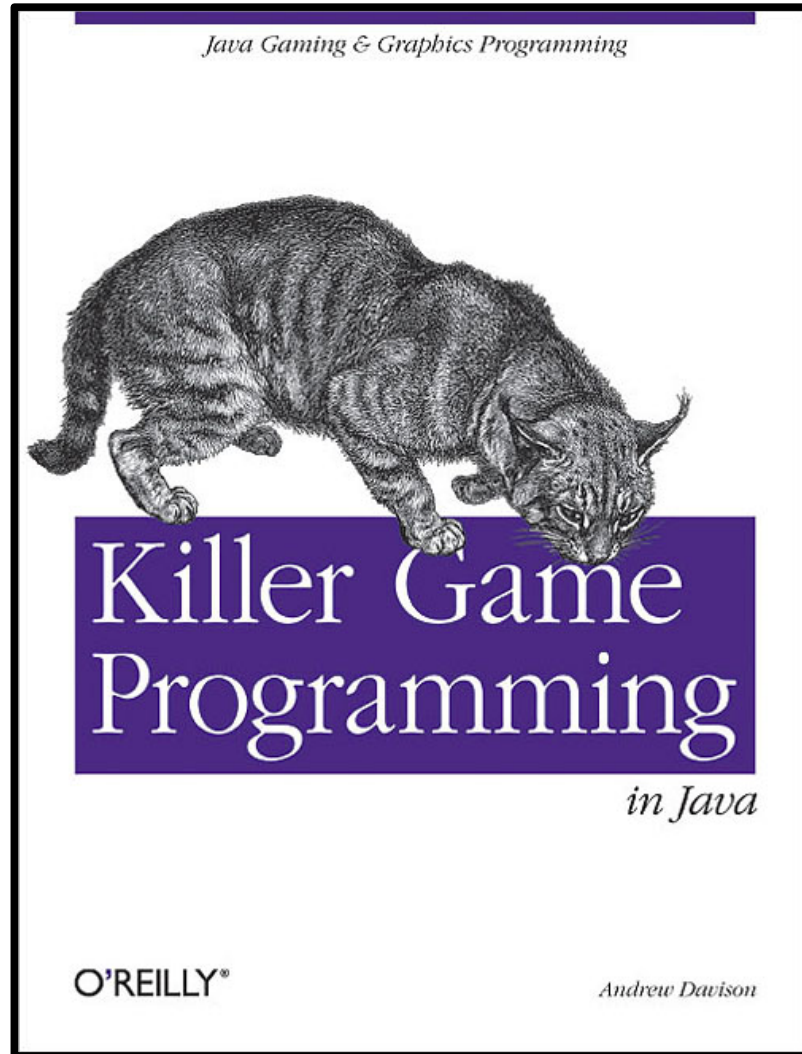


Démonstration Partie 1

Boucle et affichage

- 1) Boucle simple
- 2) Séparation game et moteur
- 3) ajout d'un repaint

- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
- Réseau



- Populous pas adapté aux machines actuelles



Boucle simple

- Première boucle

```
public class Princ1 {  
  
    static Game g;  
    // prog principal  
    public static void main(String[] args) {  
        // creation du jeu  
        g = new Game();  
        // appel à la boucle  
        boucle();  
    }  
  
    // boucle de jeu  
    public static void boucle() {  
        while (g.fini == false) {  
            g.update();  
            g.render();  
        }  
    }  
}
```

Boucle simple (2)

- Code de l'afficheur

Boucle

```
// boucle de jeu
public static void boucle() {
    while (g.fini == false) {
        g.update();
        g.render();
    }
}
```

Game

```
public void render() {
    p.repaint();
}
```

JPanel

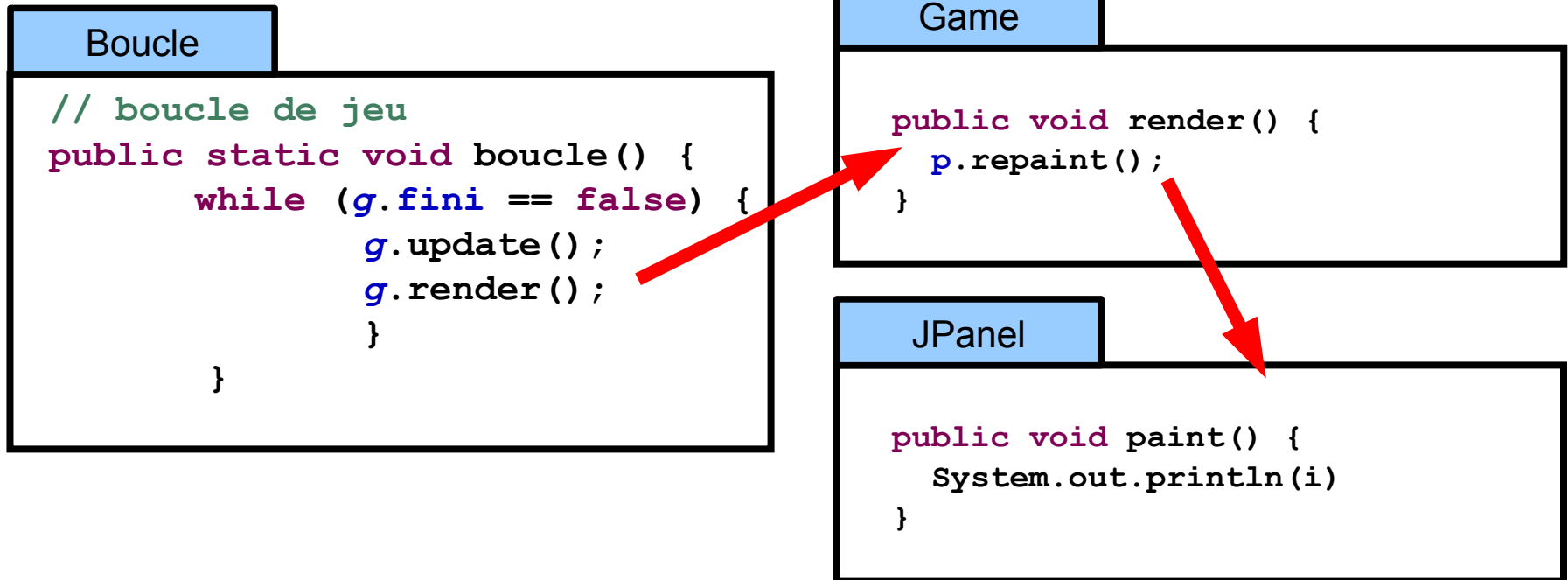
```
public void paint() {
    System.out.println(i)
}
```

- Problème

1033, 4210, 6245 , 6453, 6632,6810,,6986,7161,7345, 7526,7705,
7880, 8063, 8085, 8284, 8508, 8729, 8951, 9175, 9399, 9621, 9842,
10001, 10001

Boucle simple (2)

- Code de l'afficheur

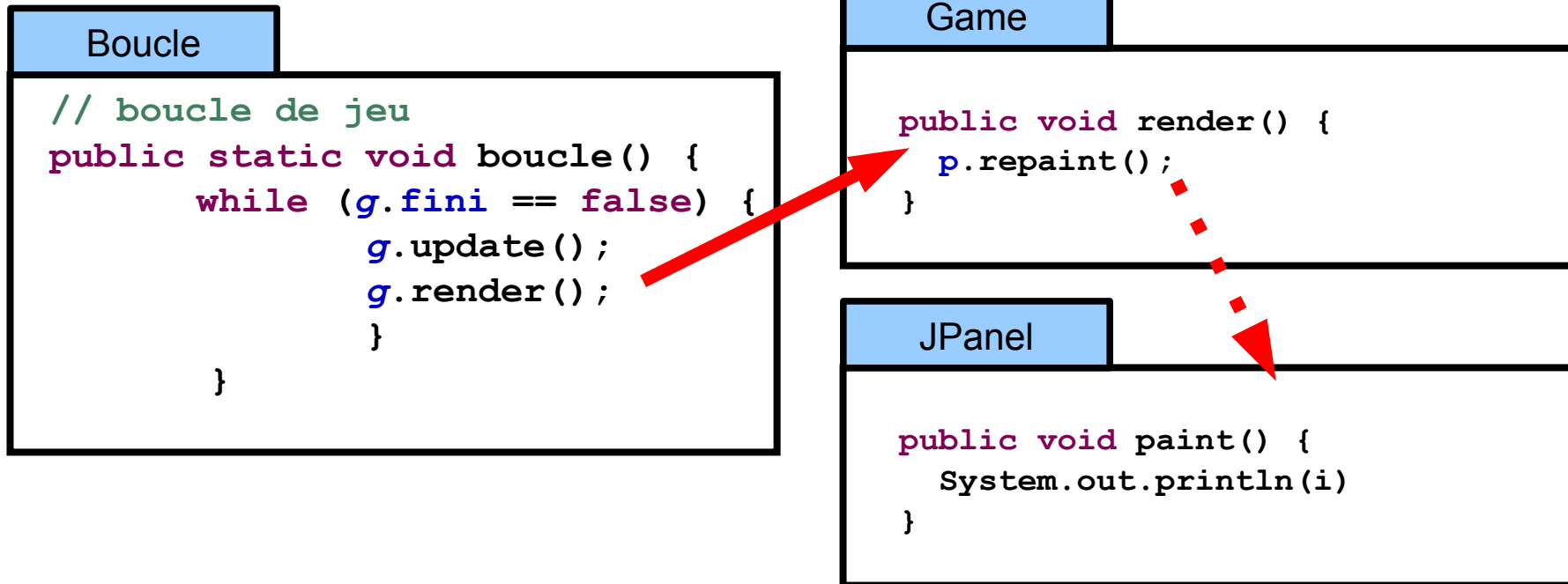


- Problème

1033, 4210, 6245 , 6453, 6632,6810,,6986,7161,7345, 7526,7705,
7880, 8063, 8085, 8284, 8508, 8729, 8951, 9175, 9399, 9621, 9842,
10001, 10001

Boucle simple (2)

- Code de l'afficheur

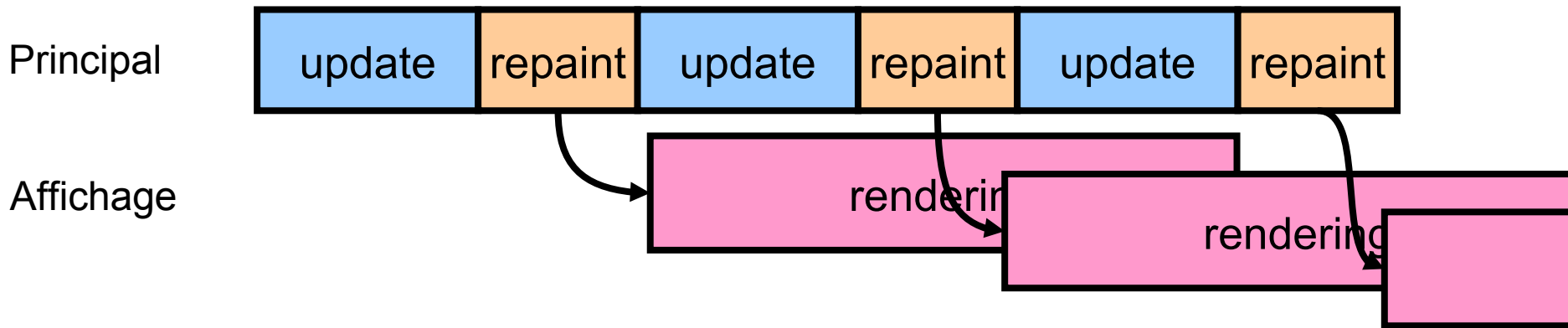


- Problème

1033, 4210, 6245 , 6453, 6632,6810,,6986,7161,7345, 7526,7705,
 7880, 8063, 8085, 8284, 8508, 8729, 8951, 9175, 9399, 9621, 9842,
 10001, 10001

- **Origine du problème**

- Repaint transmet la **demande** d'affichage
- Repaint fusionne la demande

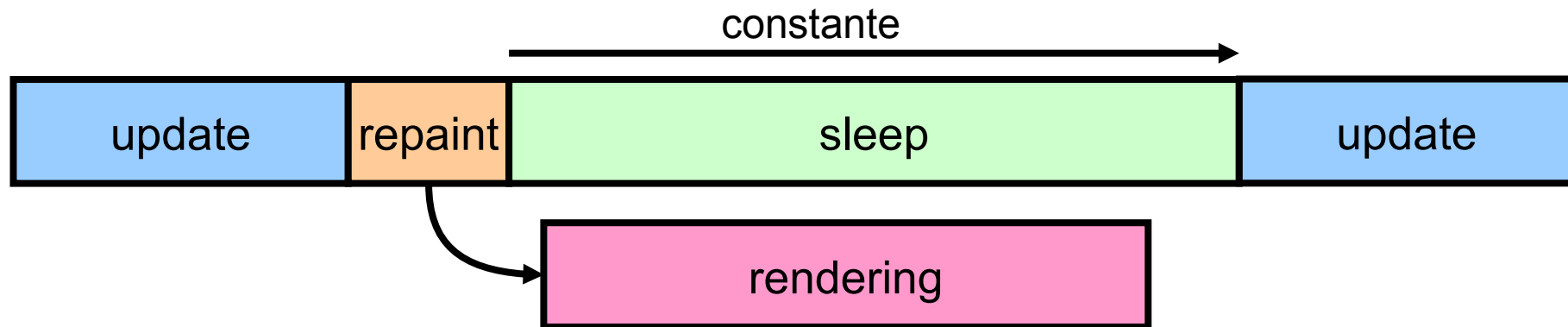


- **Solution**

- Pause
- Rendre la main à l'affichage

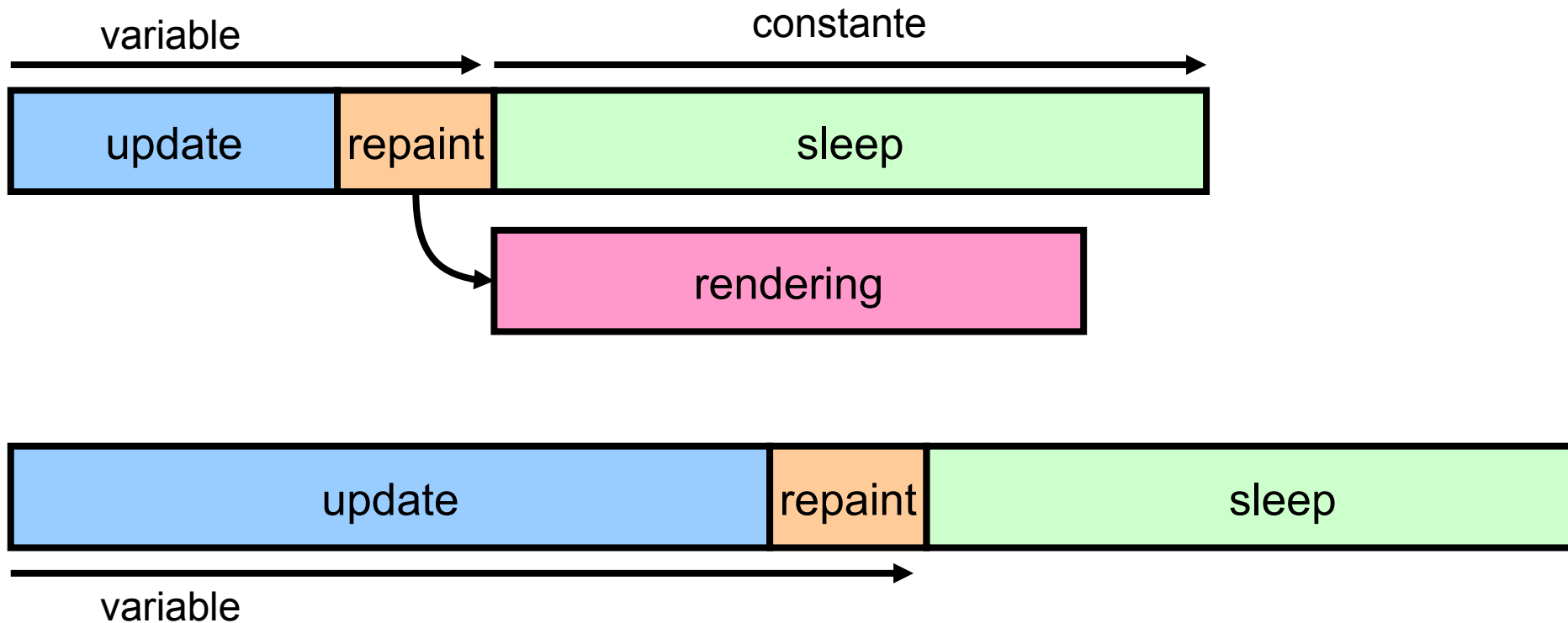
Boucle + sleep

- Redonner la main à l'affichage (Méthode sleep)

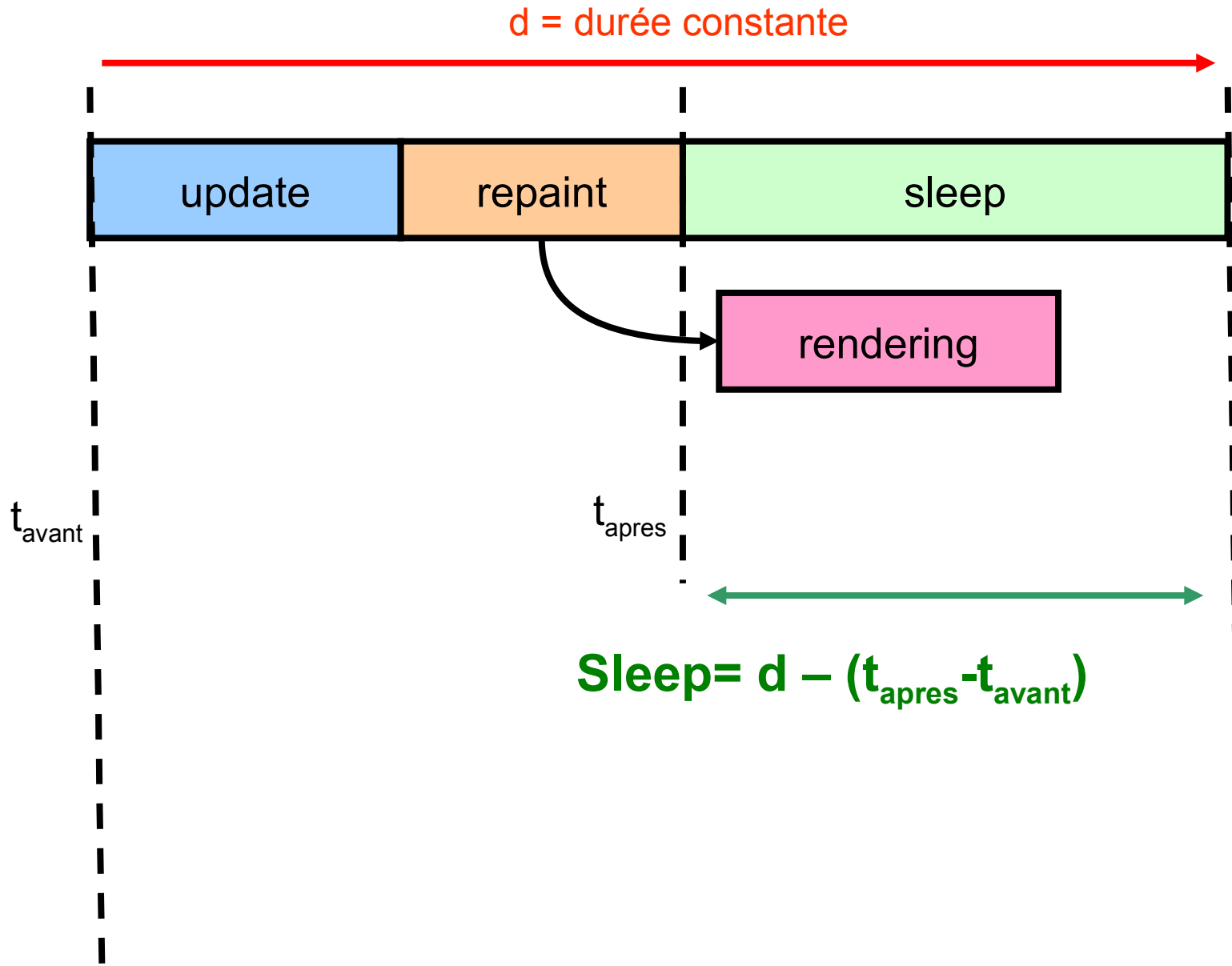


```
static void boucle() throws InterruptedException {  
  
    while (g.fini == false) {  
        g.update();  
        g.render();  
        // rendre la main  
        Thread.sleep(10);  
    }  
}
```

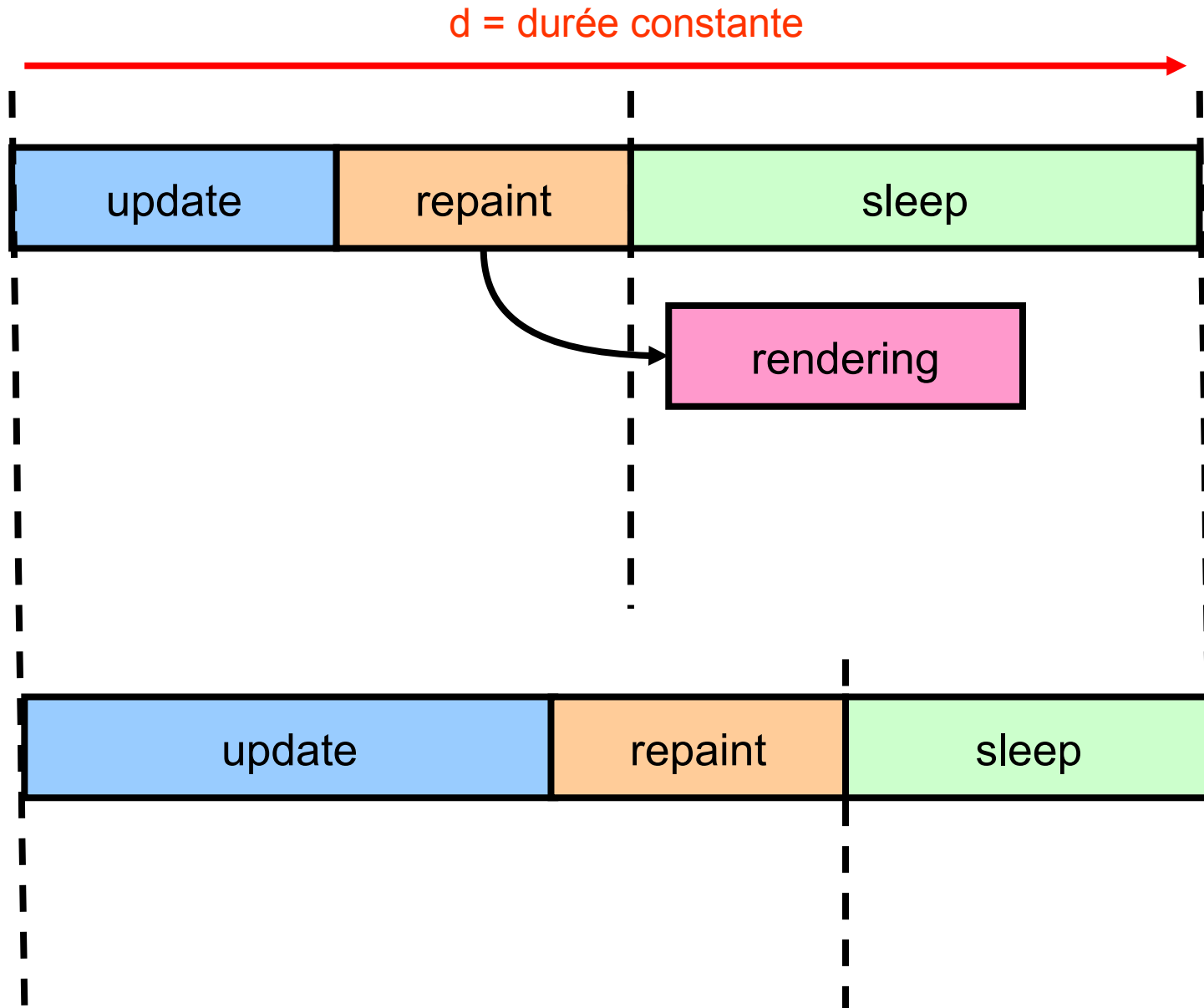
- **Résultat**
 - Toutes les itérations s'affichent
- **Problème: durée variable**
 - En fonction de la machine, du moment et des calculs



Boucle + sleep adaptatif



Boucle + sleep adaptatif (2)



- mettre à jour la duree du sleep

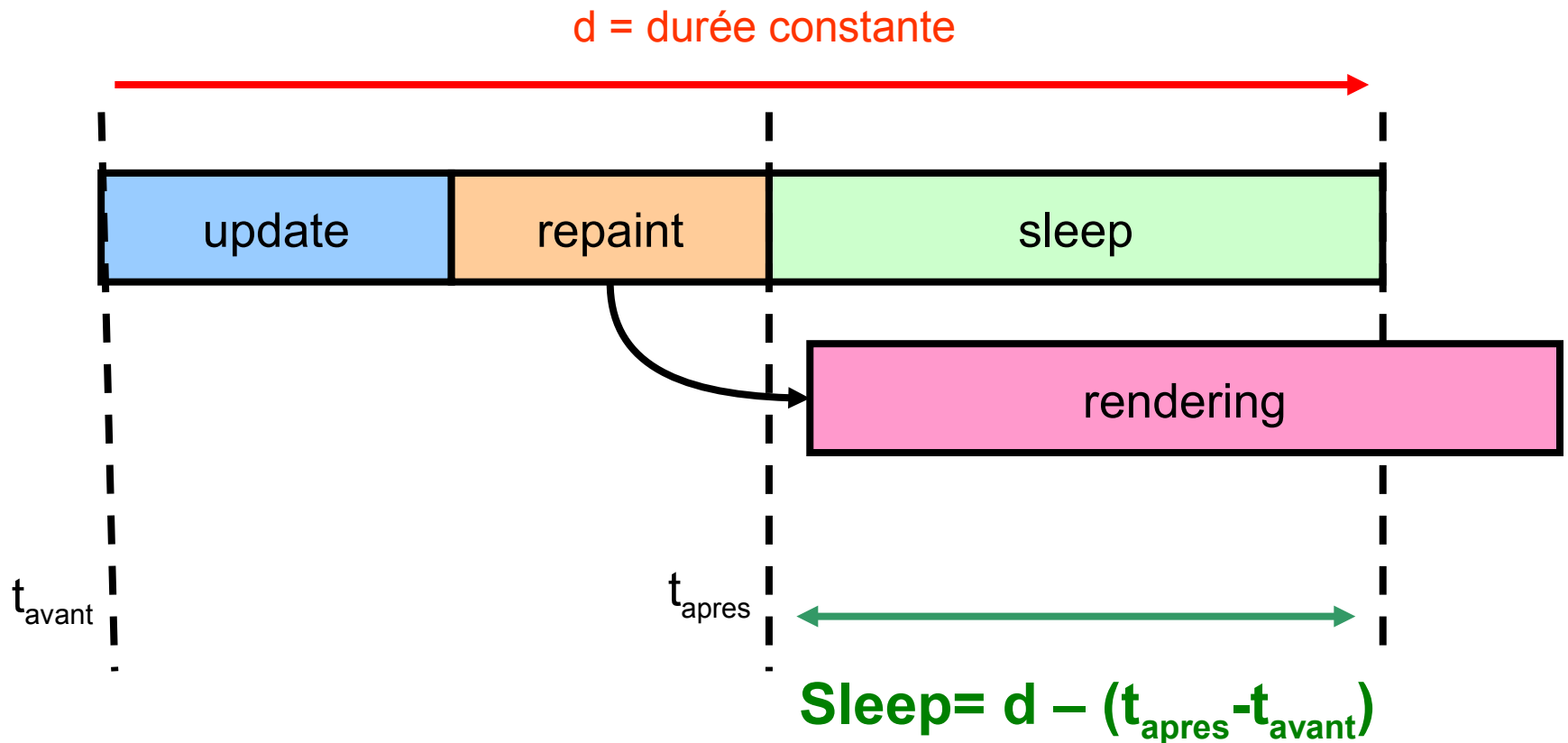
```
static void boucle() throws InterruptedException {
    // duree de la boucle
    long duree = 10;
    while (g.fini == false) {
        // recupere temps avant
        long avant = System.currentTimeMillis();

        g.update();
        g.render();
        // recupere temps après
        long apres = System.currentTimeMillis();

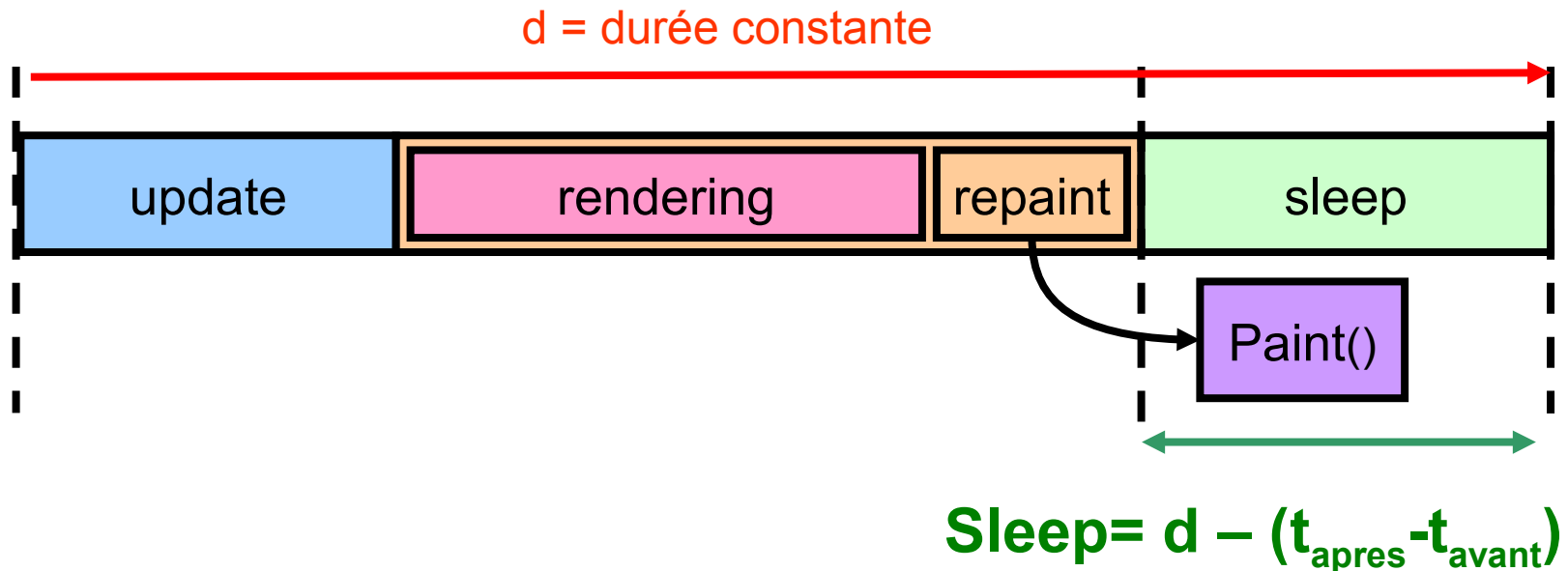
        // rendre la main
        Thread.sleep(duree - (apres - avant));
    }
}
```

- **Problème**

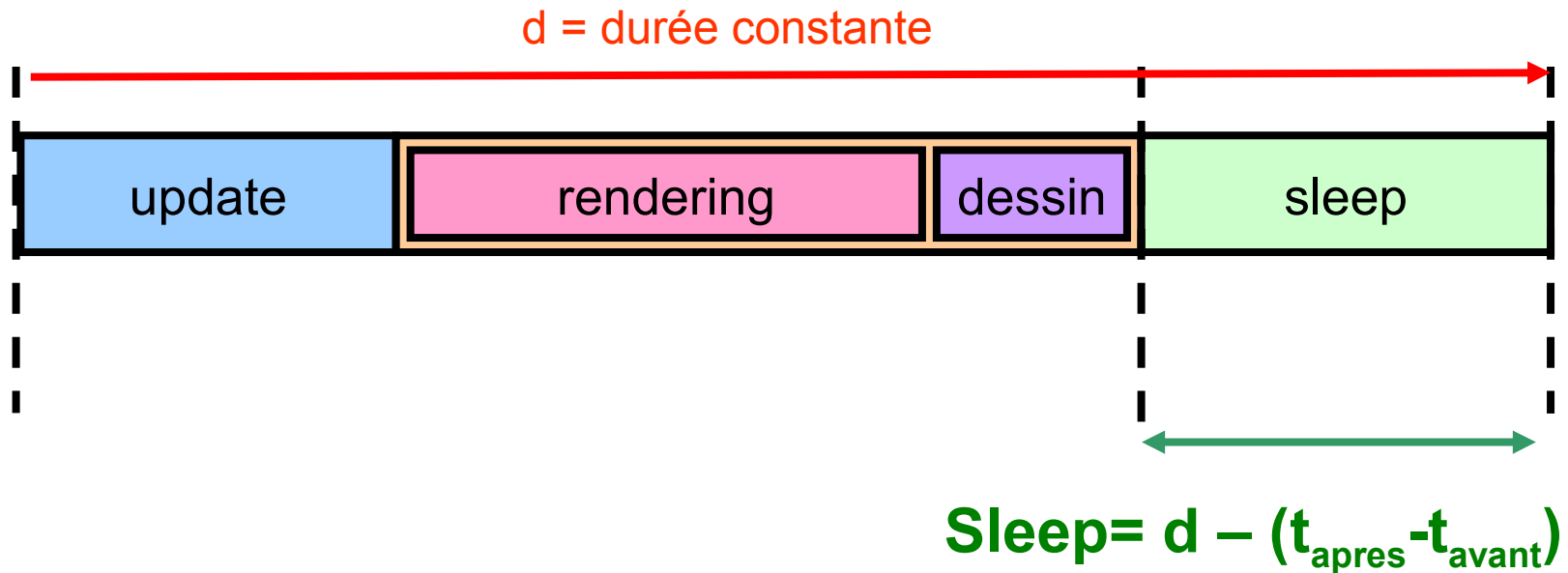
- La durée du rendu n'est pas prise en compte



- Intégrer rendering dans thread principal
 - Repaint rapide (ex : double buffering)



- Intégrer rendering dans thread principal
 - Repaint rapide (ex : double buffering)
 - Méthode qui crée l'image (active rendering)



Active rendering & double buffering (2)

Cas idéal



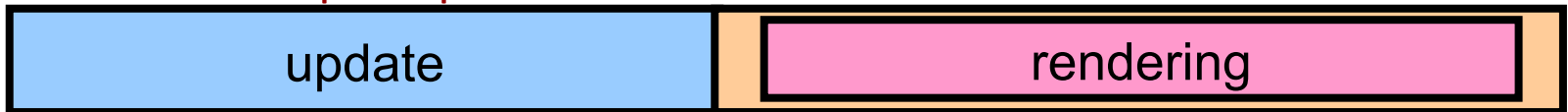
Pb1 – précision du sleep



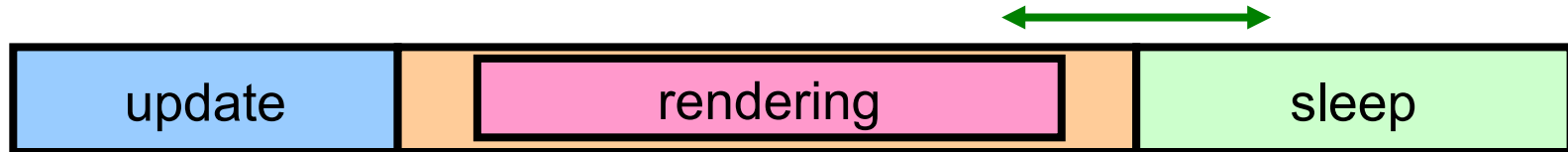
Pb2 – précision du timer



Pb3 – pas assez de temps disponible



- **FPS**
 - Nombre de frame par secondes
 - Dans l'absolu, quelque chose proche de 60
- **Problemes de temps**
 - Précision du timer
 - Précision du sleep
 - Tache trop lourde



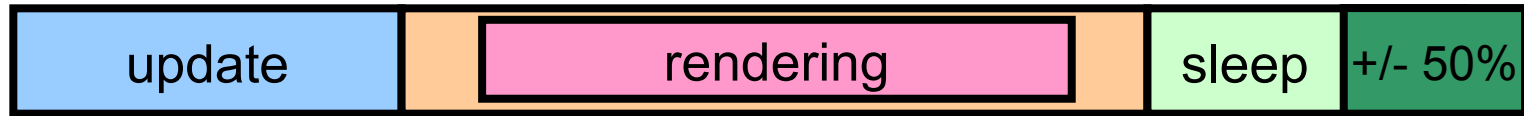
- Le timer a une certaine résolution

```
private static void sysTimeResolution() {  
    long total, count1, count2;  
    count1 = System.currentTimeMillis();  
    count2 = System.currentTimeMillis();  
    while (count1 == count2)  
        count2 = System.currentTimeMillis();  
    total = 1000L * (count2 - count1);  
    System.out.println(total);  
}
```

Résultat → 16000

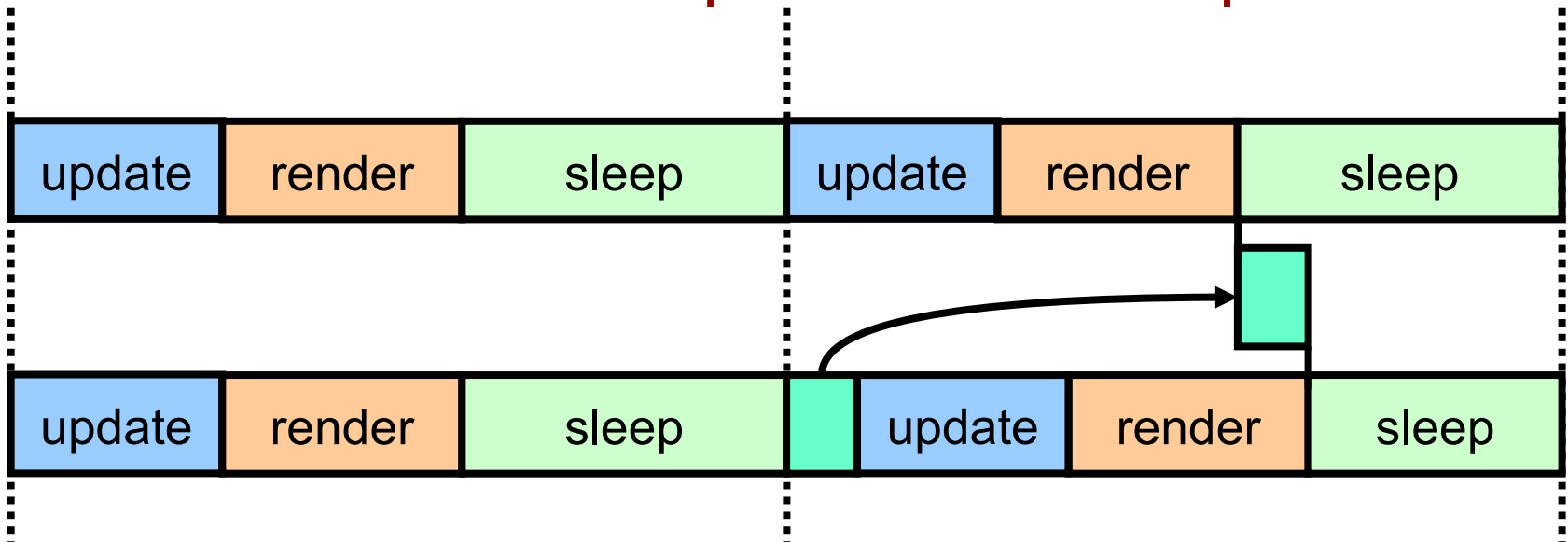
- Le timer ne peut pas distinguer 0 de 15 ms
 - Le sleep sera tout le temps de durée (ex 10ms)
 - L'itération durera entre 10 et 25 ms
- Solution
 - Utiliser la classe **Perf**

Pb2: Précision du sleep

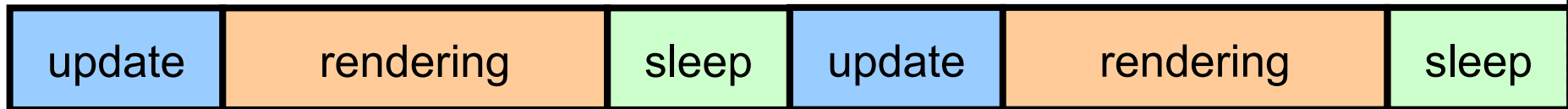


- **Durée méthode sleep**
 - avec des erreurs (de 1% à 20%)

- **Retenir le delai et le prendre en compte**



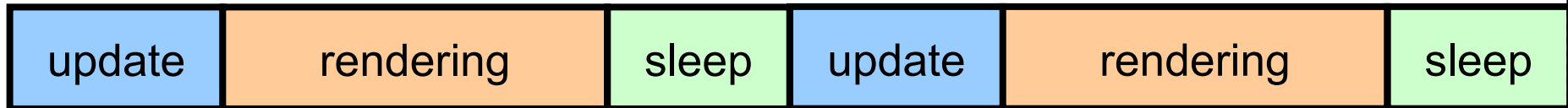
- Cas normal



- Cas problématique



- Cas normal

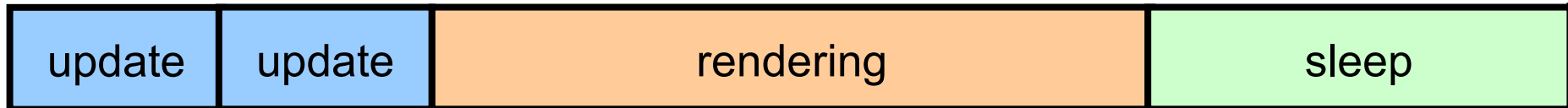


- Cas problématique



- Solution

- Faire plusieurs update pour un affichage
- Le jeu reste aussi rapide mais moins fluide



Démonstration Partie 2

Gestion du temps

02x01 – wait simple

02x02 – wait et affichage

02x03 – wait adaptatif

- Moteur générique

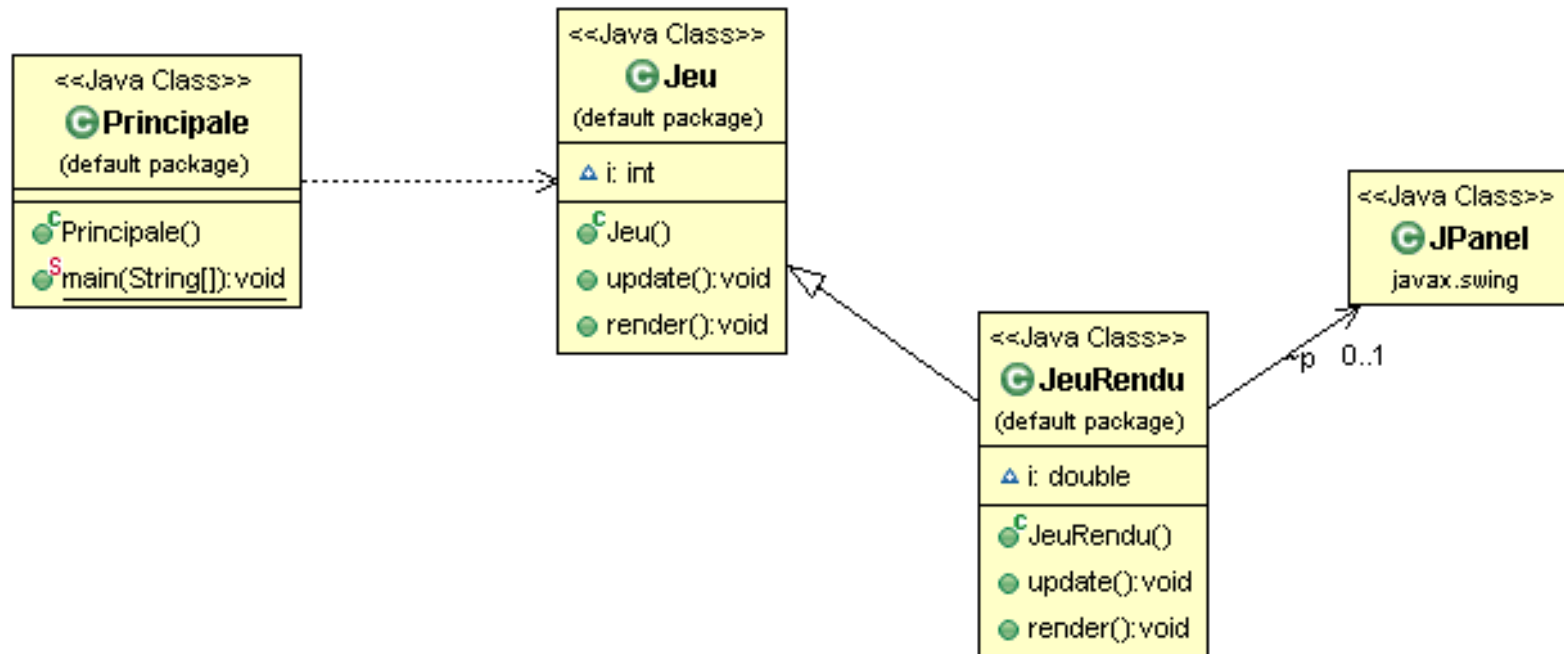
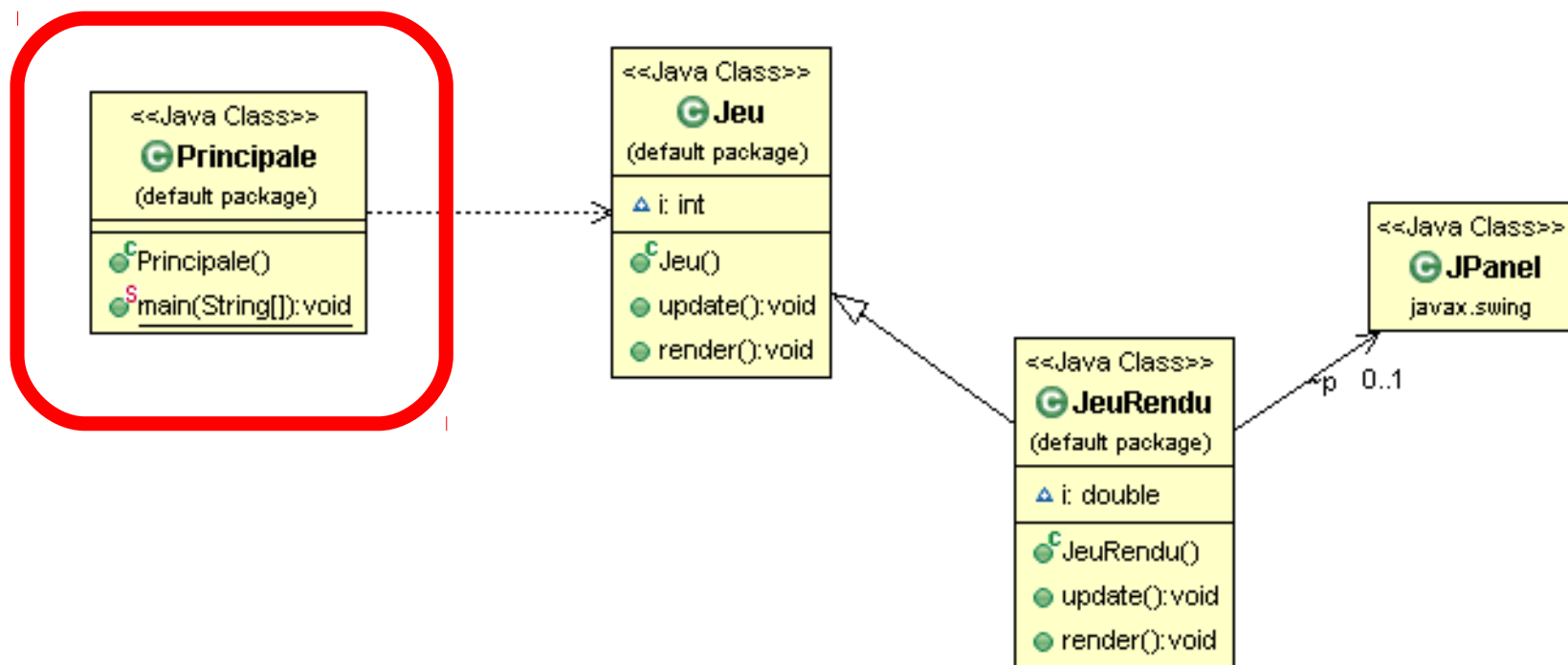


Diagramme de classe

- Moteur générique



- **Classe Thread**

- Erreur pas à 20 % mais 100
- À la place de 9ms
 - 5 ms ou 17 ms



- **Si on est à 17 ms**

- 58 Fps

- **API Java**

- « *subject to the precision and accuracy of system timers and schedulers* »

Sauf que ...

```
for (int i = 0; i < n; i++) {  
    j.update();  
    j.render();  
  
    // apres le render en nanos  
    long timafter = System.nanoTime();  
  
    while (System.nanoTime() - beforeTime - dureeBoucle * 1000L < 0) {  
  
    }  
  
    // avant la prochaine boucle  
    beforeTime = System.nanoTime();  
}
```


Sauf que ...

```
for (int i = 0; i < n; i++) {  
    j.update();  
    j.render();  
  
    // apres le render en nanos  
    long timafter = System.nanoTime();  
  
    while (System.nanoTime() - beforeTime - dureeBoucle * 1000L < 0) {  
  
    }  
  
    // avant la prochaine boucle  
    beforeTime = System.nanoTime();  
}
```

- On tourne dans le vide

Sauf que ...

```
for (int i = 0; i < n; i++) {
    j.update();
    j.render();

    // apres le render en nanos
    long timafter = System.nanoTime();

    // duree en nanos
    long duree = dureeBoucle * 1000L - (timafter - beforeTime);
    System.out.println("doit attendre" + duree / 1000L);

    // sleep en millisecond
    if (duree < 0)
        throw new AssertionError("trop de temps");
    System.out.println("duree attendue" + duree);
    while (System.nanoTime() - beforeTime - dureeBoucle * 1000L < 0) {
    }

    beforeTime = System.nanoTime();
    System.out.println("duree réelle attente" );
    System.out.println((beforeTime - timafter)/ 1000L + "\n");
}
```

Démonstration Partie 2

Gestion du temps

02x04 – attente sans sleep

- Définir une boucle de jeu
 - Séparer rendu / mise à jour
- Assurer FPS constant
 - Évaluer retard
 - Mise en attente adaptée
- Reste
 - Quoi dans update ?
 - Quoi dans render ?

Exemple de jeu

Classe Principale

```
public class Princl {  
  
    static Game g;  
    // prog principal  
    public static void main(String[] args) {  
        // creation du jeu  
        g = new Game();  
        // appel à la boucle  
        boucle();  
    }  
  
    // boucle de jeu  
    public static void boucle() {  
        while (g.fini == false) {  
            g.update();  
            g.render();  
        }  
    }  
}
```

Moteur générique

```
public class Game {  
  
    int n = 0;  
    JPanel p;  
    boolean fini = false;  
  
    public void update() {  
        n = n + 1;  
        if (n > 10000)  
            fini = true;  
    }  
  
    public void render() {  
        p.repaint();  
    }  
}
```

Exemple de jeu

Classe Principale

```
public class Princl {  
  
    static Game g;  
    // prog principal  
    public static void main(String[] args) {  
        // creation du jeu  
        g = new Game();  
        // appel à la boucle  
        boucle();  
    }  
  
    // boucle de jeu  
    public static void boucle() {  
        while (g.fini == false) {  
            g.update();  
            g.render();  
        }  
    }  
}
```

Fixe

```
public class Game {  
  
    int n = 0;  
    JPanel p;  
    boolean fini = false;  
  
    public void update() {  
        n = n + 1;  
        if (n > 10000)  
            fini = true;  
    }  
  
    public void render() {  
        p.repaint();  
    }  
}
```

Variable

- Moteur générique

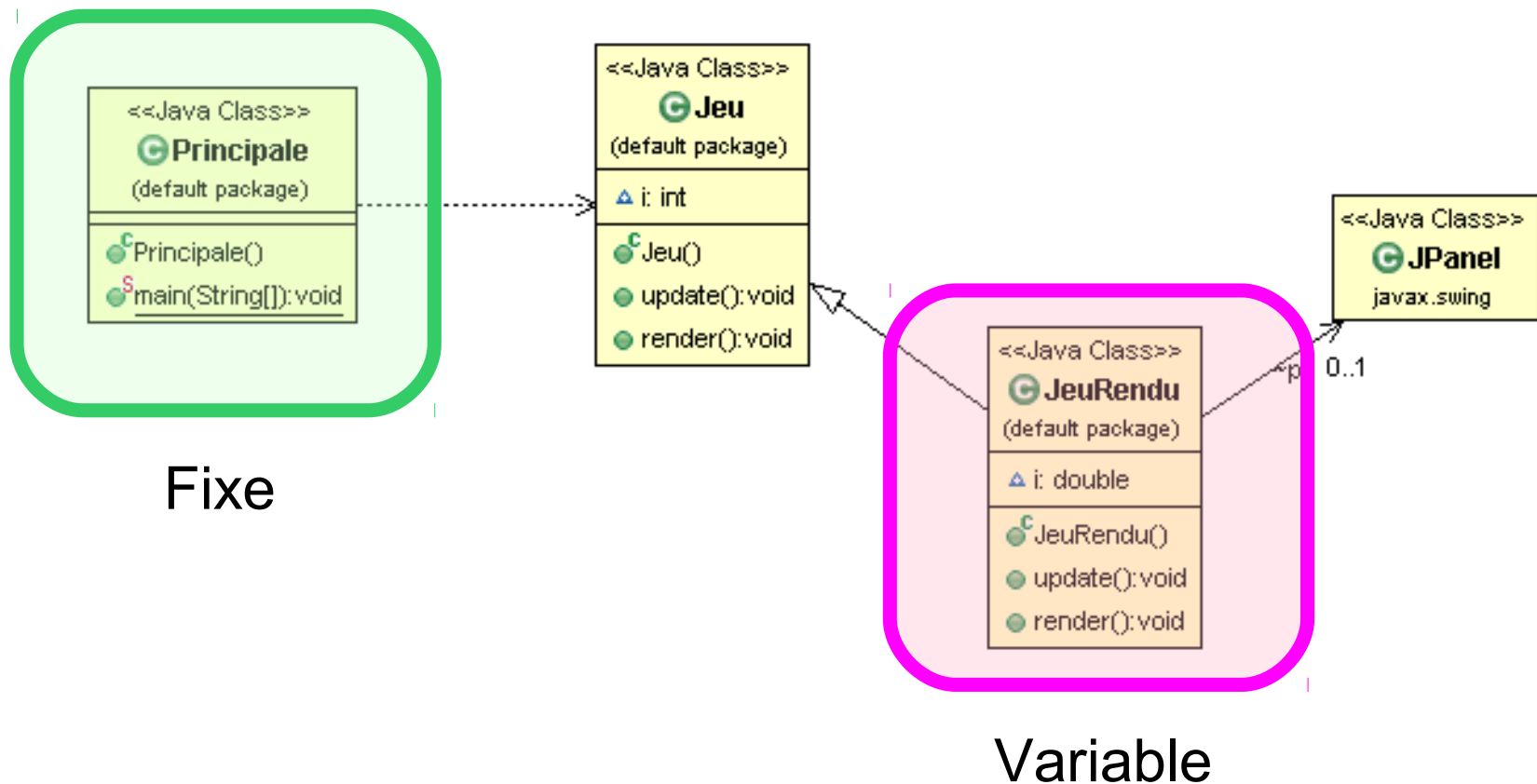


Diagramme de classe

- Séparer données et affichage

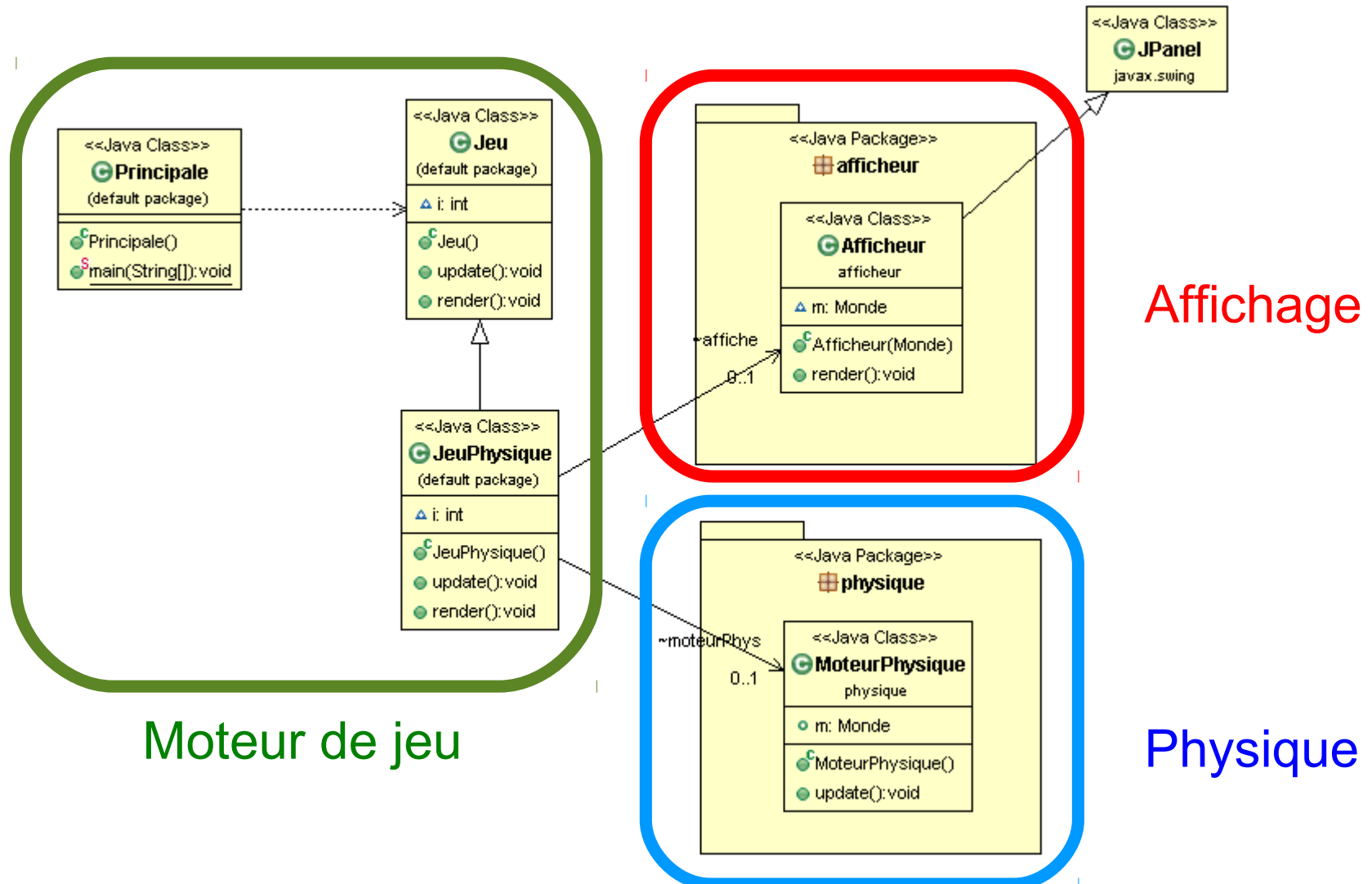
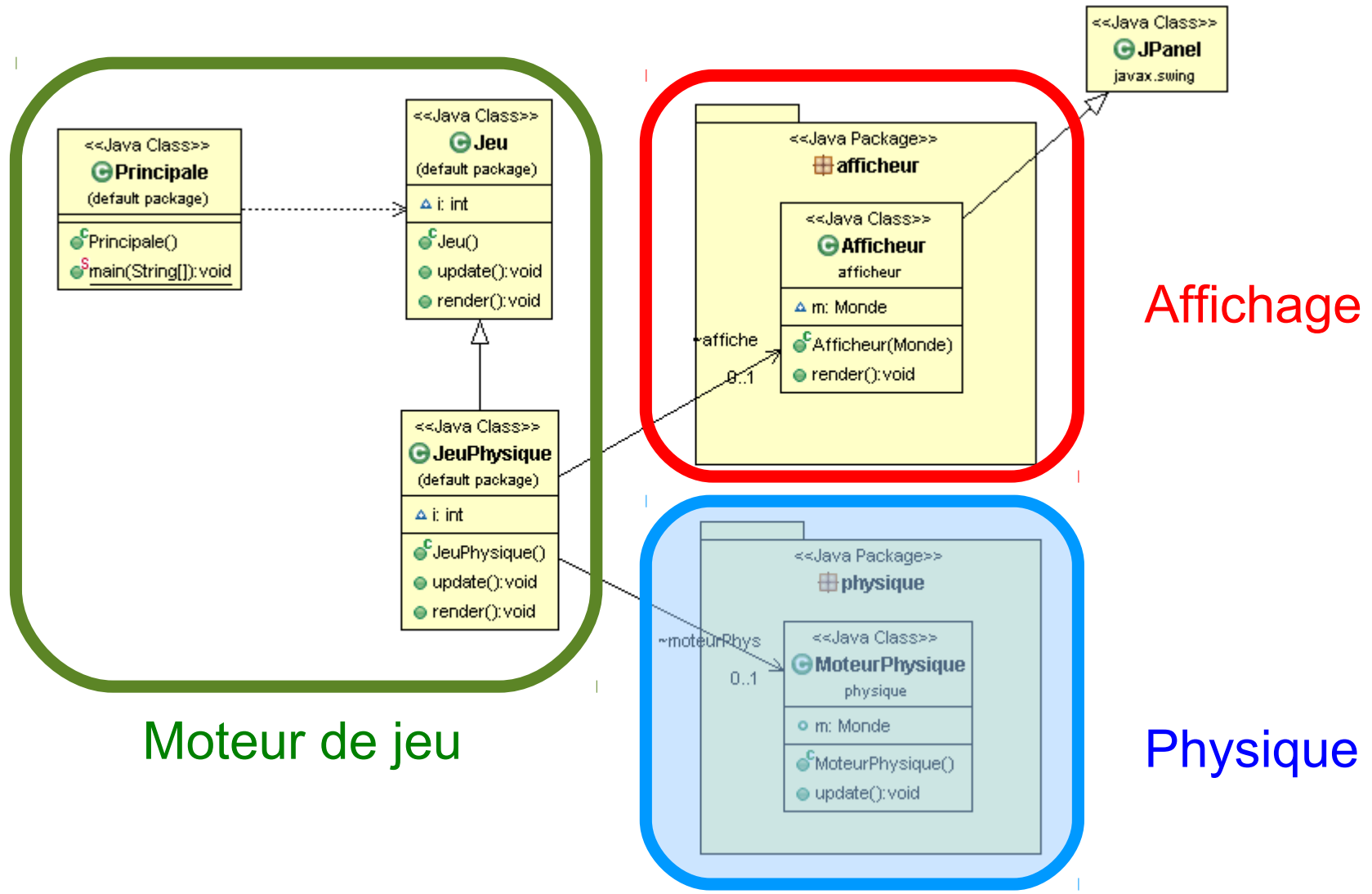


Diagramme de classe

- Séparer données et affichage



- Boucle de jeu
- Gestion du temps
- **Modèle de jeu**
 - Moteur physique
 - Gestion collisions
 - Intelligence Artificielle
- Gestion du Contrôleur
- Affichage
- Réseau

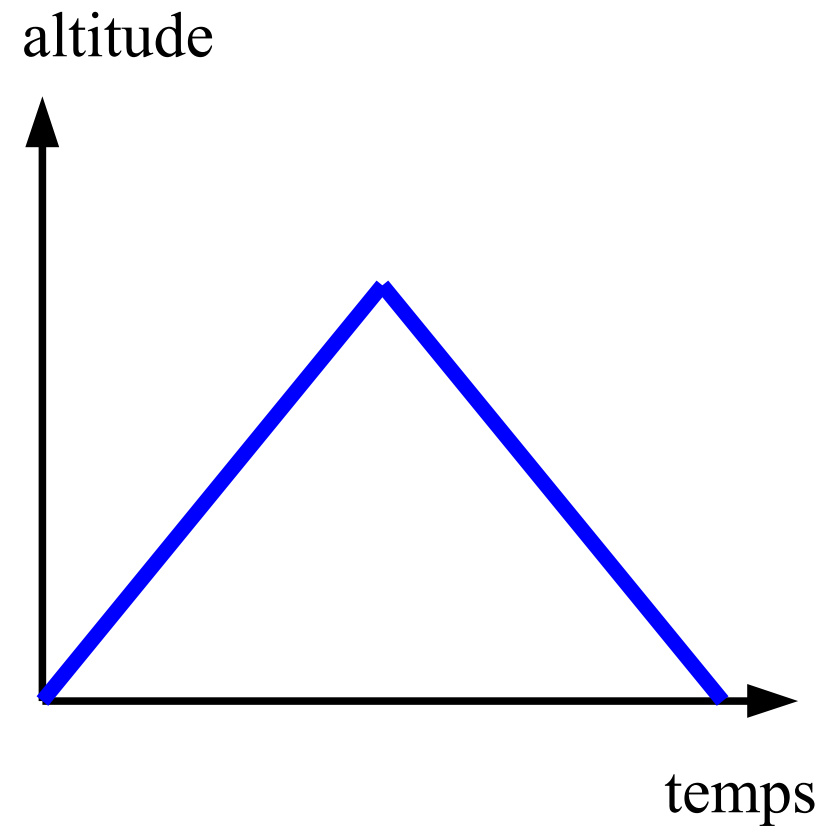
- **update()**
 - Lié au Modèle
- **Mise à jour**
 - Lois du monde et moteur physique
 - Collisions
 - Intelligence artificielle
- **Intégration des contrôles du joueur (après)**



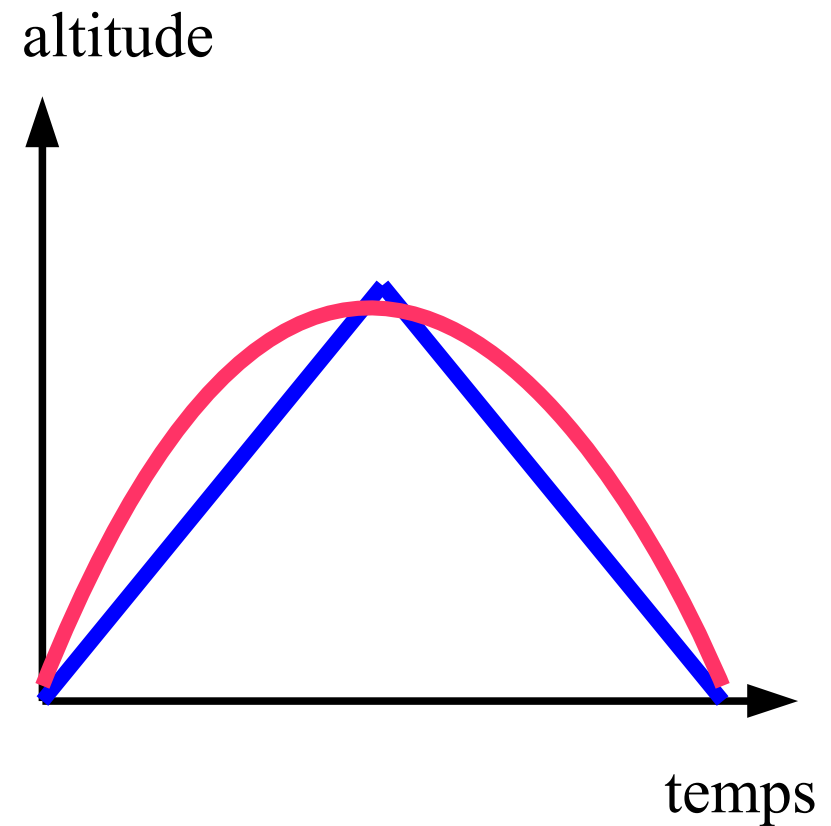
Newton vu par Gotlib

- Règles du jeu
 - A définir
- Une itération 10ms
 - Gestion des temps de déplacements
 - 1 pixel => 100 pixels / secondes
- Ressenti du joueur
 - Exemple jeu de course

- Comportement de saut

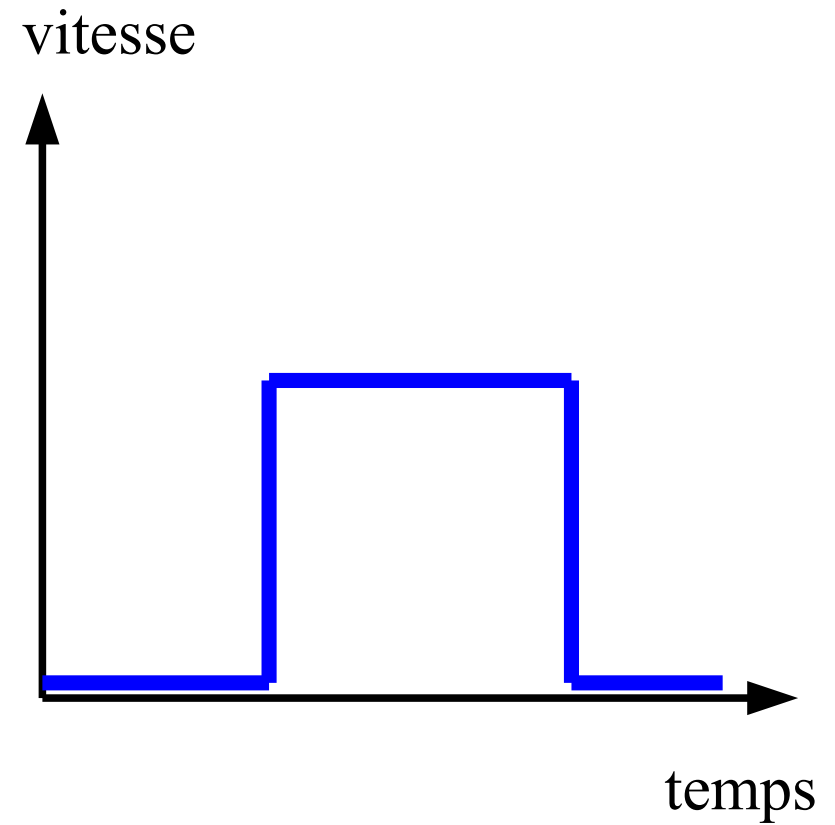


- Comportement de saut

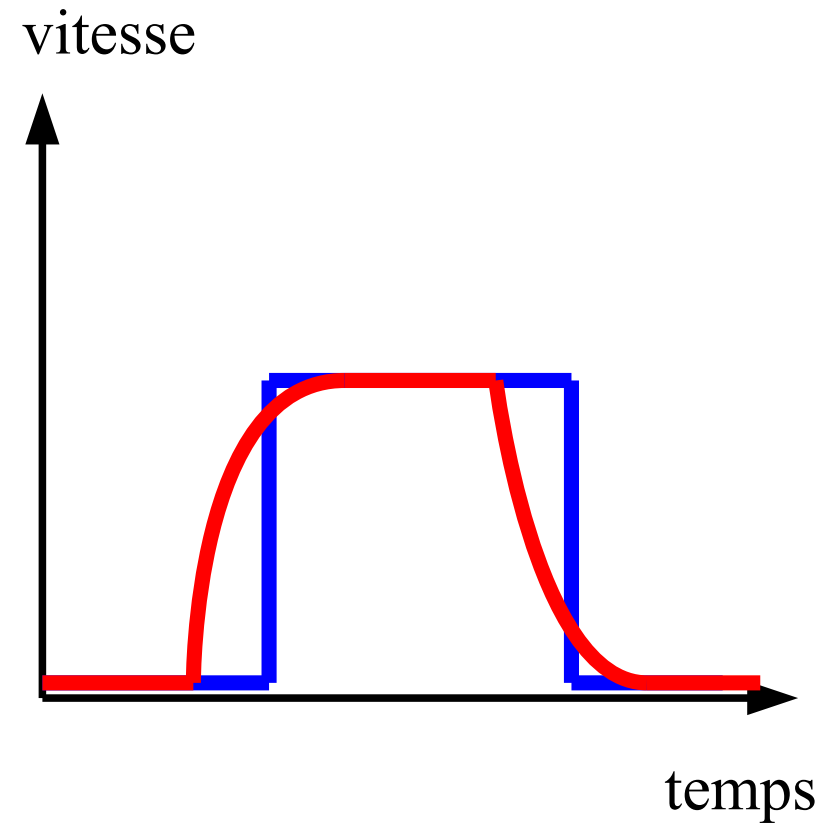


« le réalisme des jeux vidéos » (merci Dorian, jeuxvideos.com)

- Course

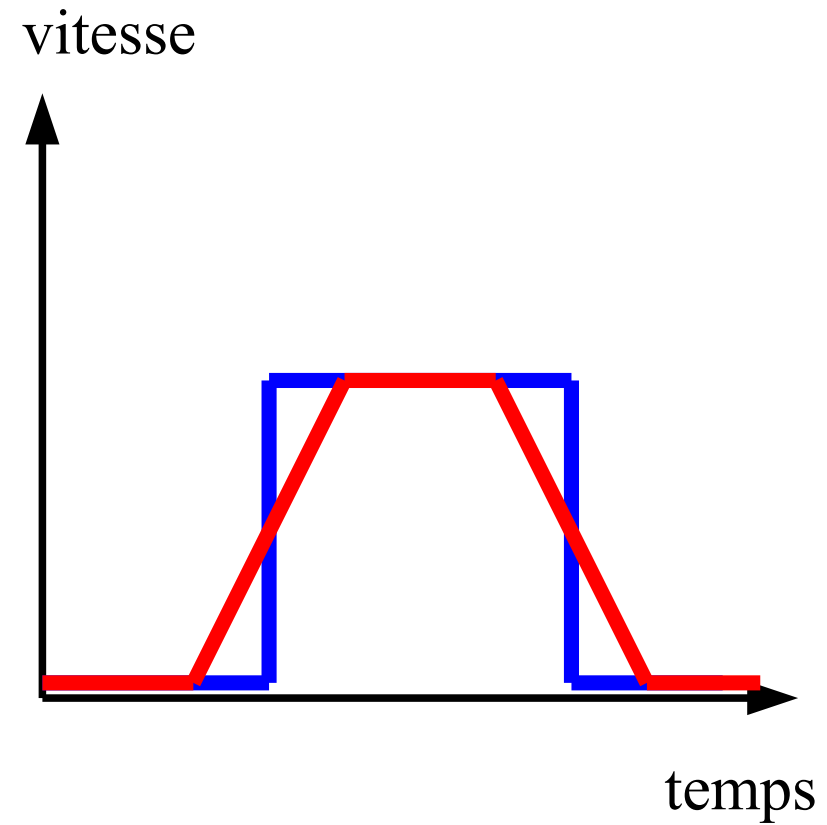


- Course



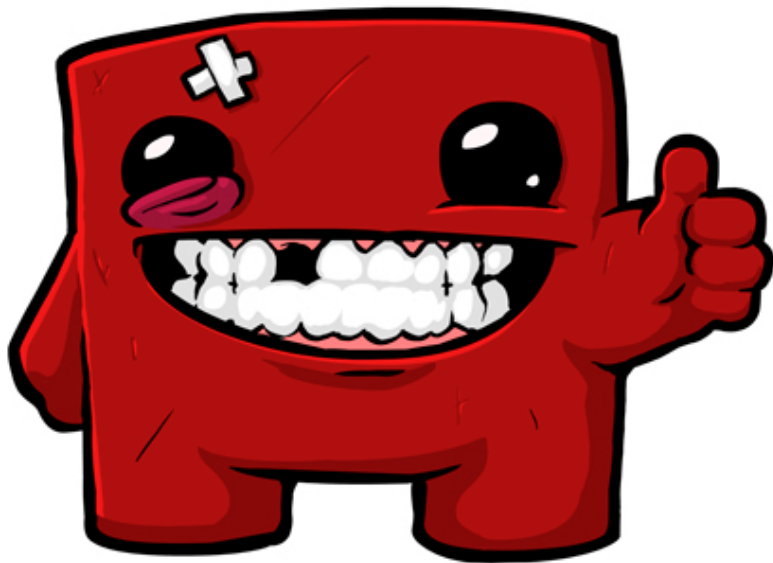
- Modèles du monde

- Course

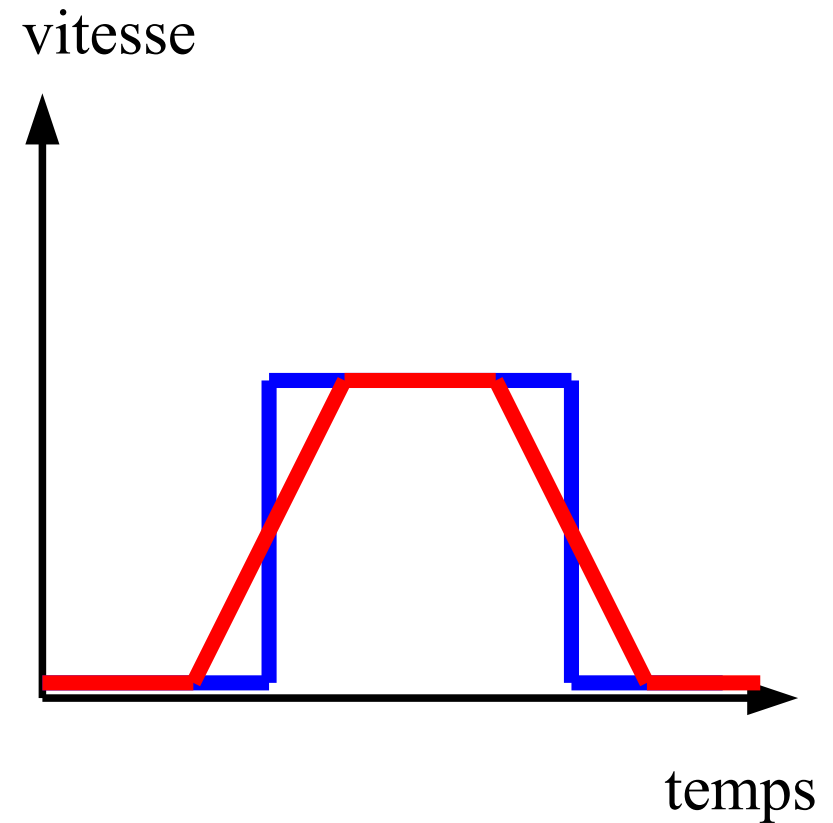


- Modèles du monde

- Course



Super meat boy - (indépendant)



- Modèles du monde

Modèles physiques

- Mécanique point

- Position
- Vitesse
- Accélération

- A chaque temps

$a = \text{fixée}$

$v = v + a \cdot dt$

$x = x + v \cdot dt$

```
// permet de modéliser un objet
public class Objet {

    // modele de l'objet
    // position
    public double px = 0;
    public double py = 0;
    // vitesse
    double vx = 0, vy = 0;
    // acceleration
    double ax = 0, ay = 0;

    // boundingbox
    double width = 10, height = 10;

    // mise à jour avec des equations physiques
    public void update() {
        px = px + vx;
        py = py + vy;
        vx = vx + ax;
        vy = vy + ay;
        //rebond sur le sol
        if (py < 0)
            vy = -vy;
    }
}
```

Démonstration Partie 3

Modèle physique

03x01 – gravité mode console

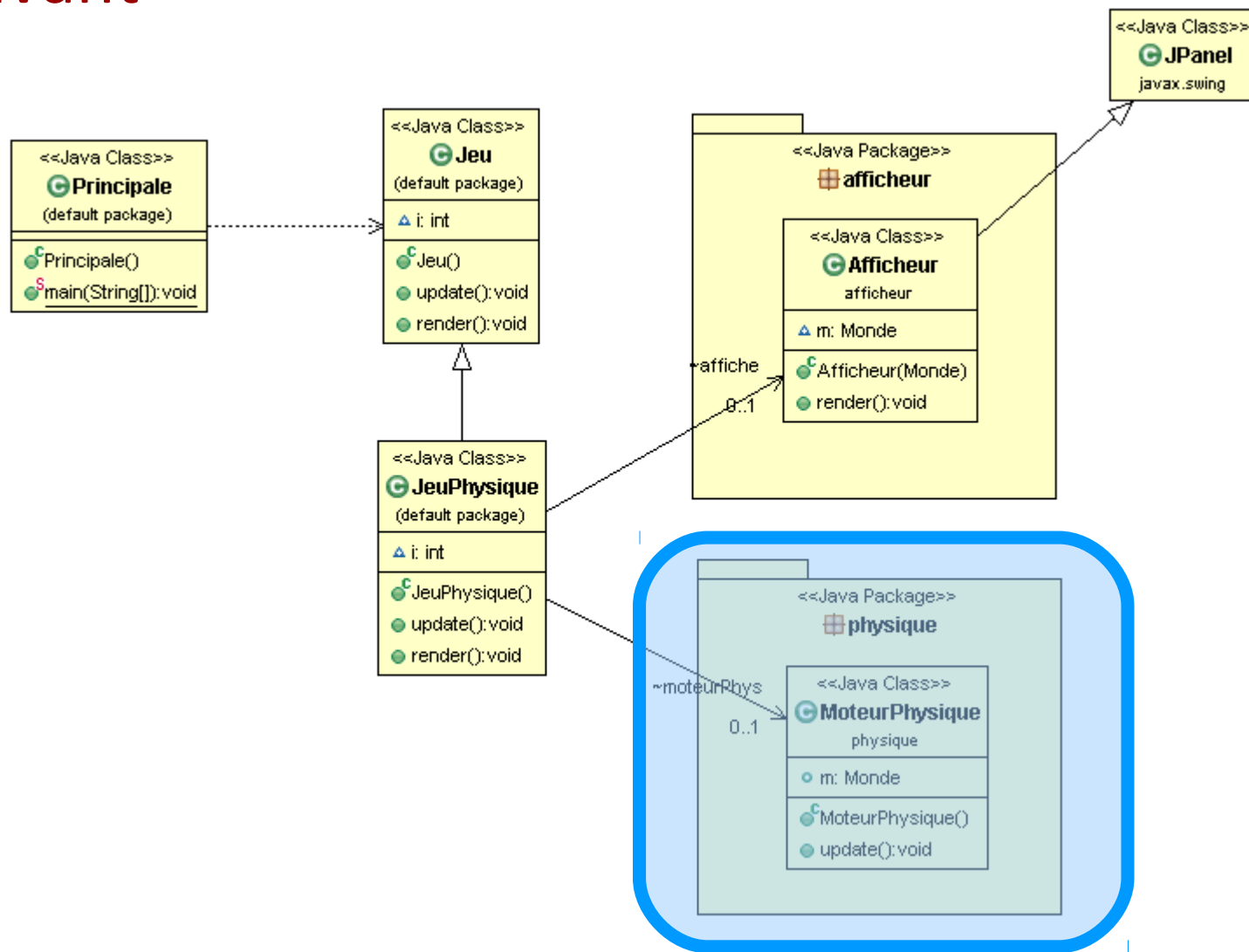
03x02 – gravité graphique / sleep cst

03x03 – gravite graphique / sleep var

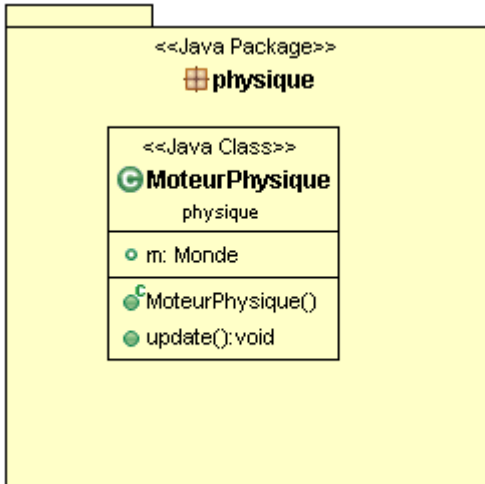
03x04 – gravite graphique / pas sleep

Diagramme de classe

- avant



- Modèle physique



- Modèle physique

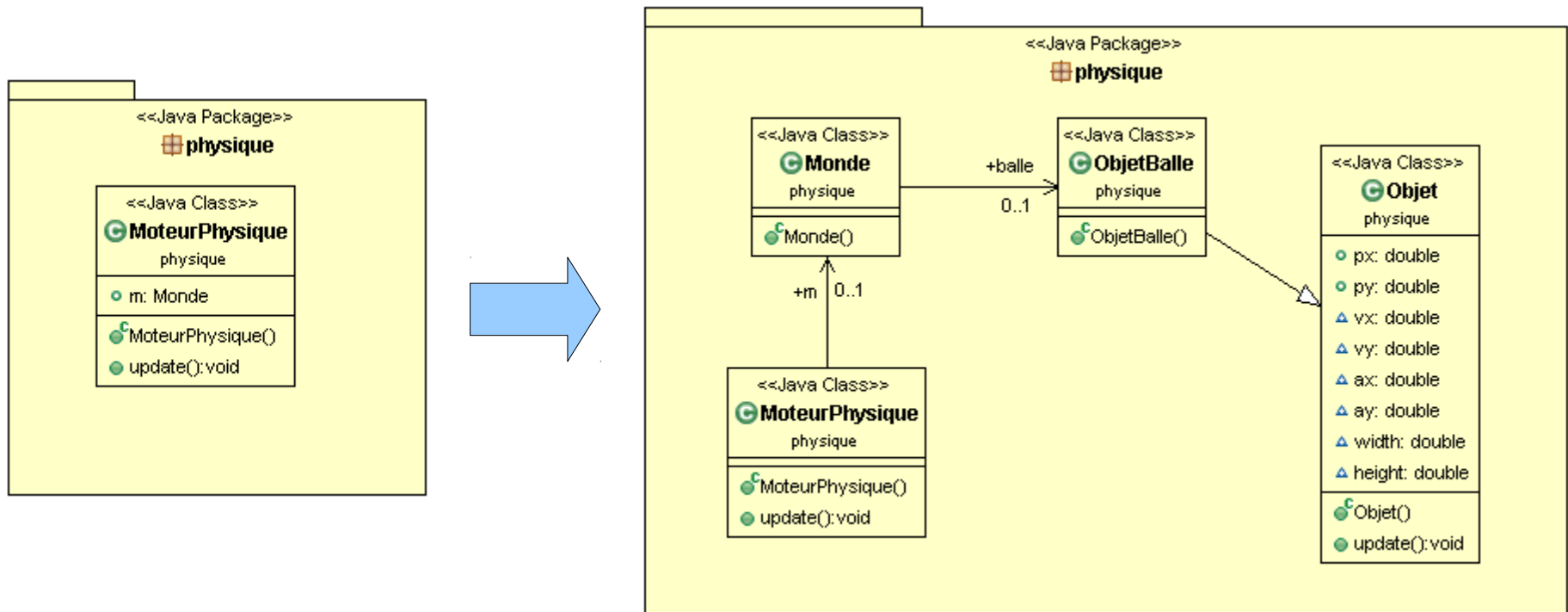


Diagramme de classe

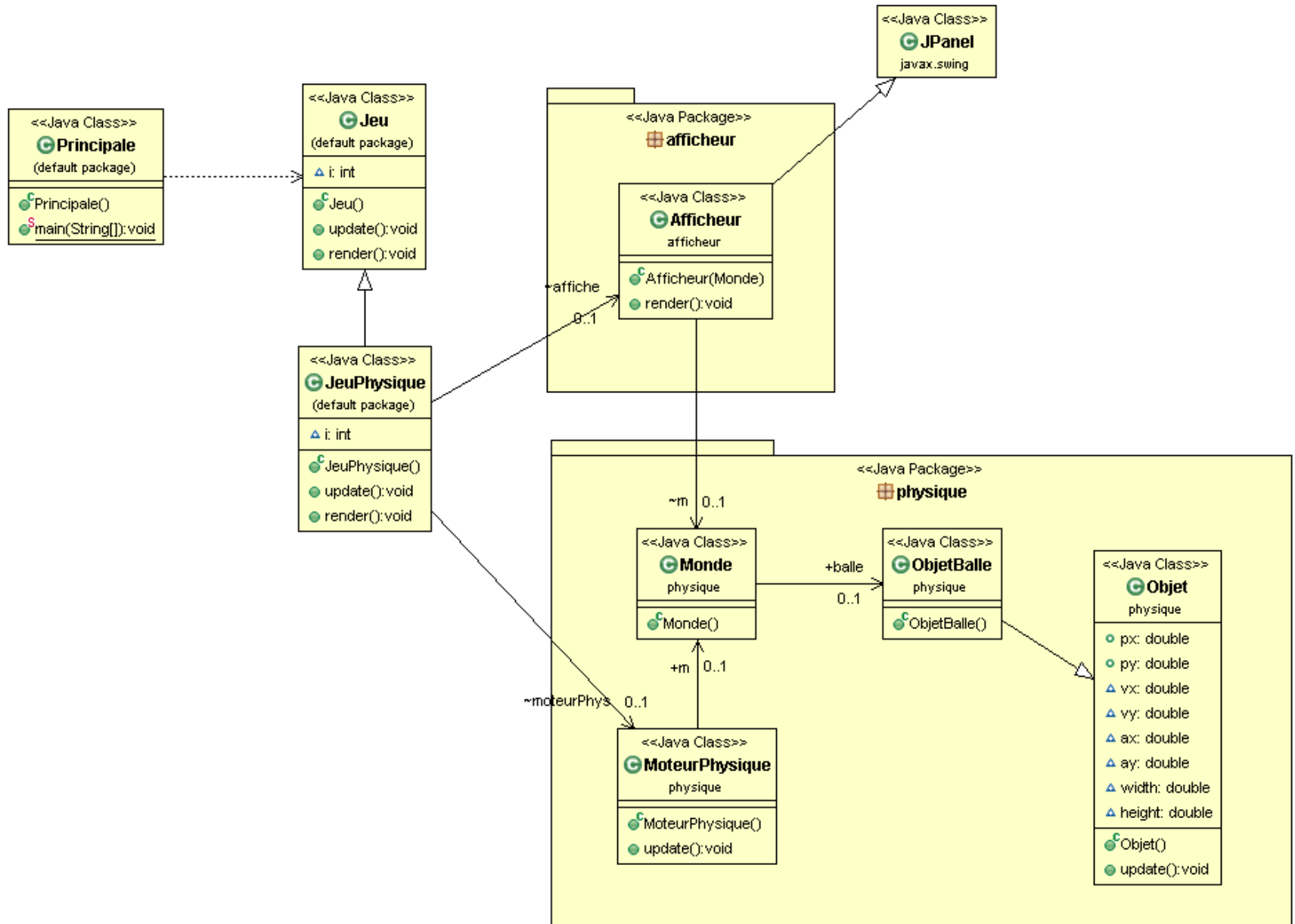
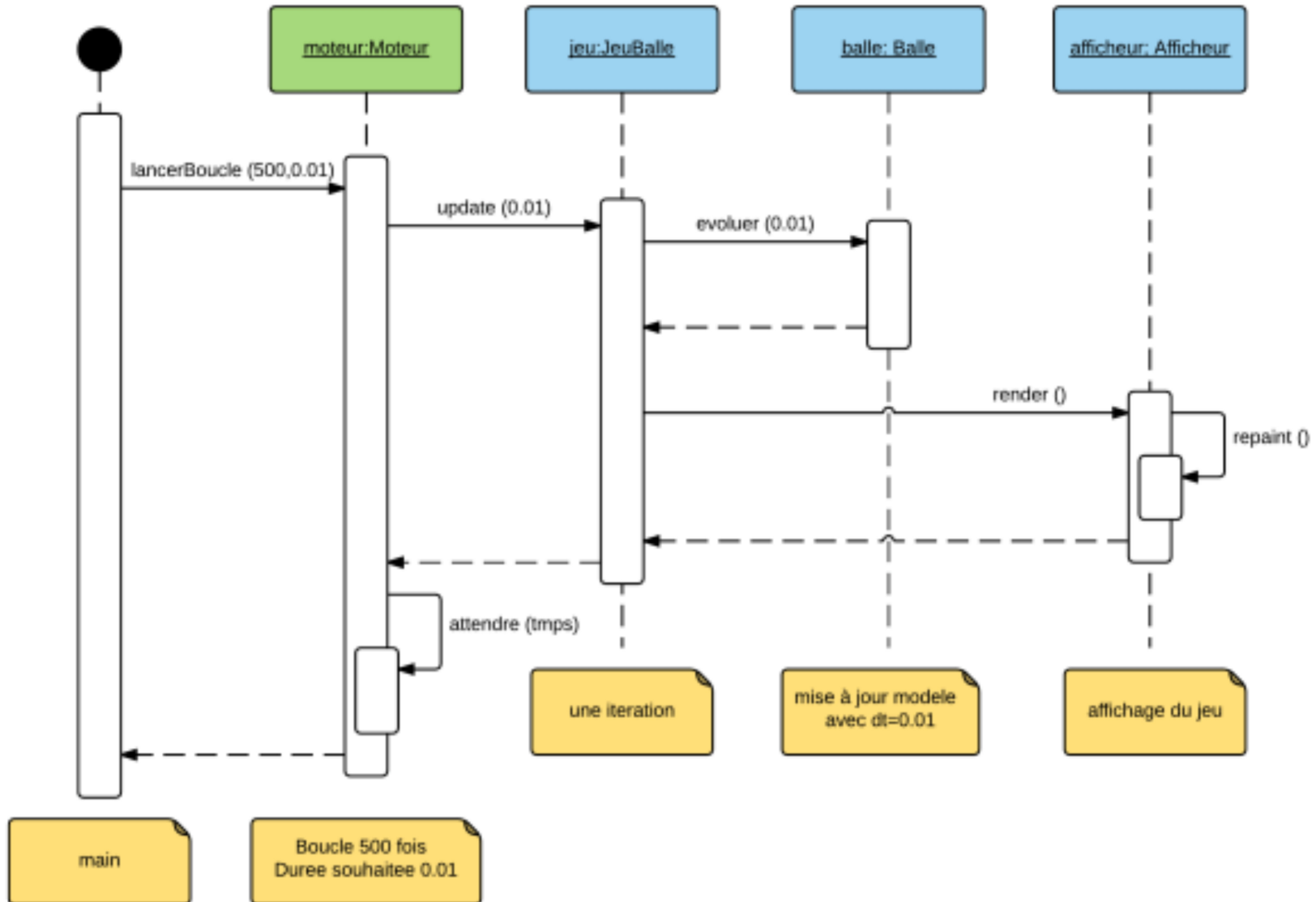
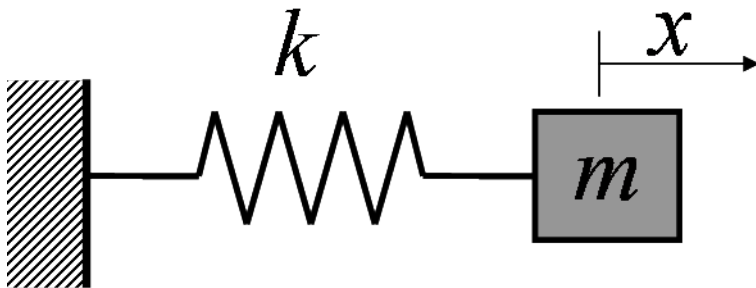


Diagramme de séquence



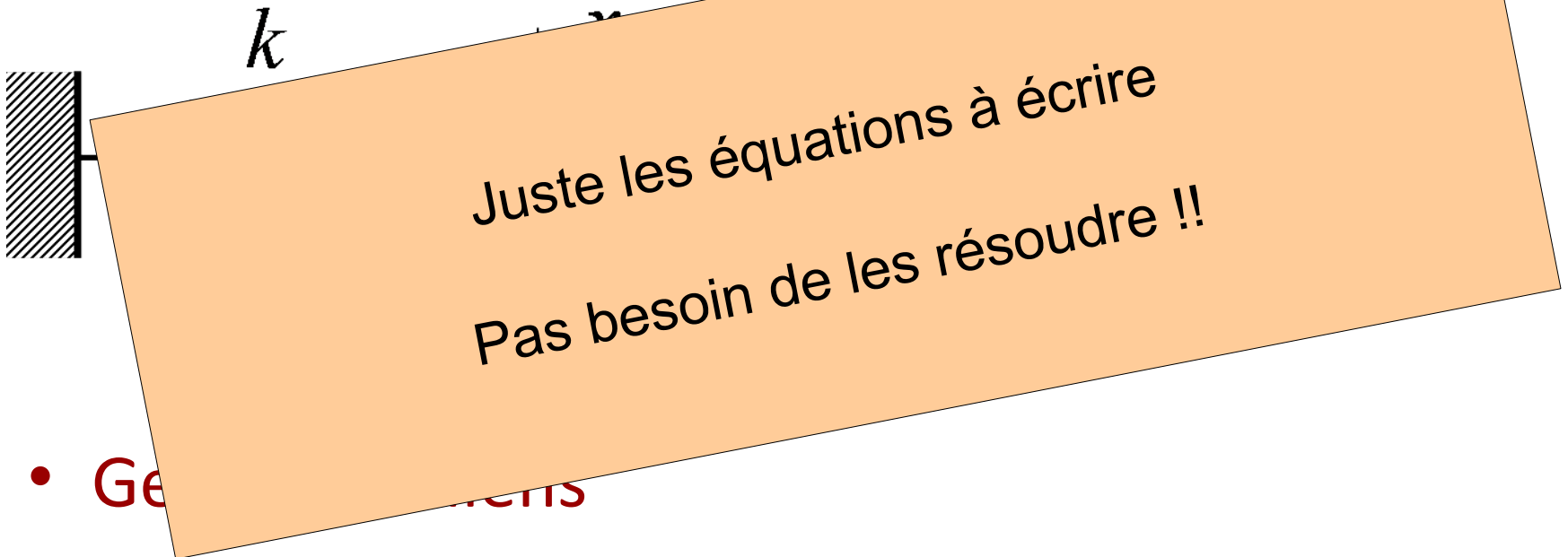
- Possibilité de faire des choses plus complexes
 - Rebond, Systemes déformables



$$F = -k(x - x_0)$$

- Gestion de liens
 - Bloc qui pend à une corde
 - Systeme de ponts

- Possibilité de faire des choses plus complexes
 - Rebond, Systemes déformables



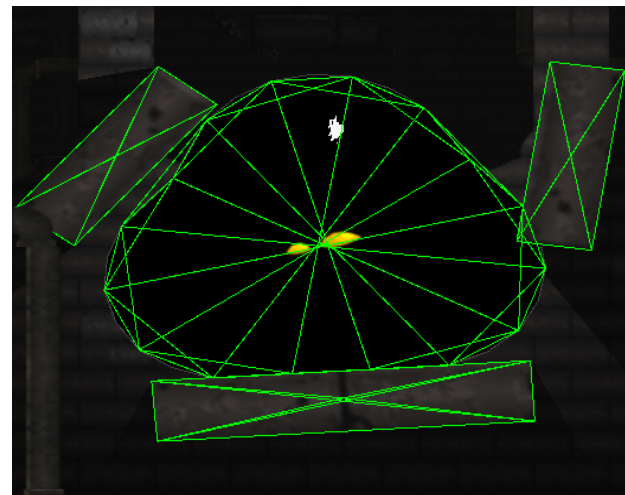
- Généralisations
 - Bloc qui pend à une corde
 - Systeme de ponts

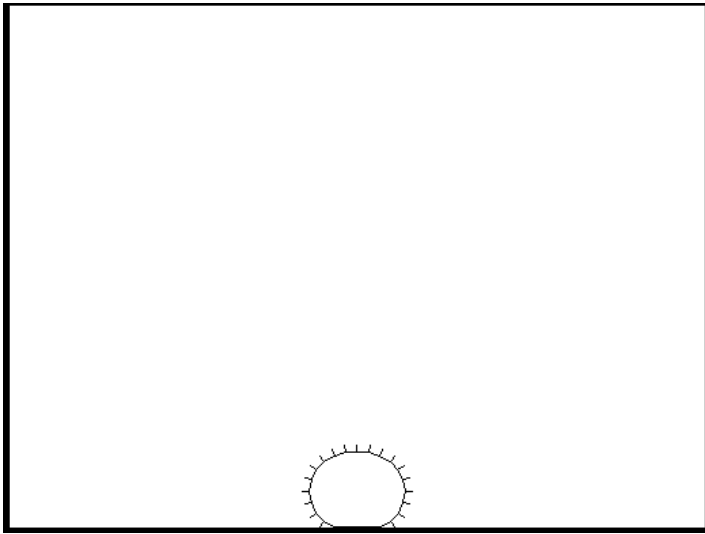
Example : World of goo



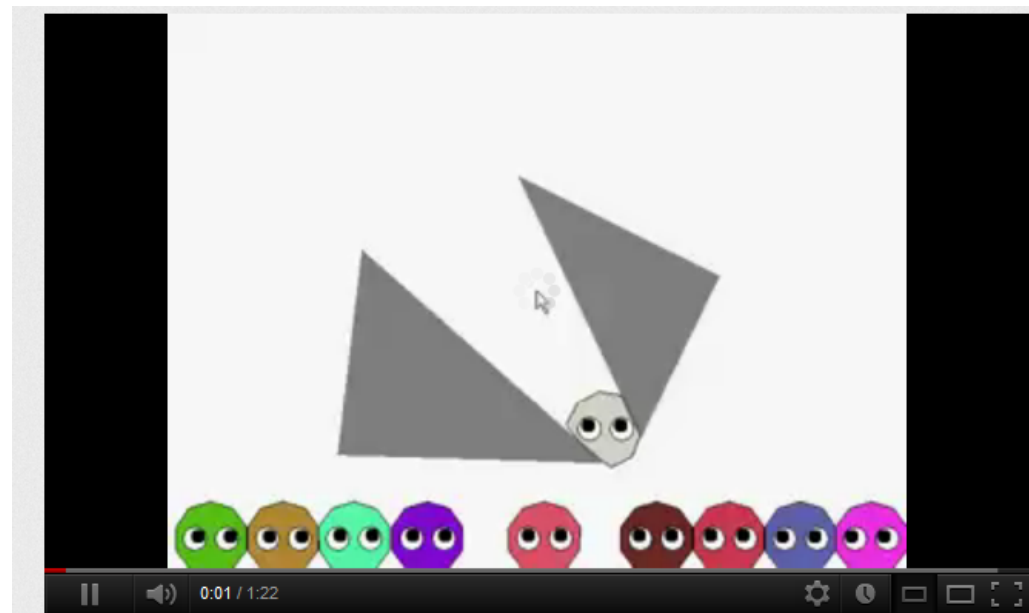


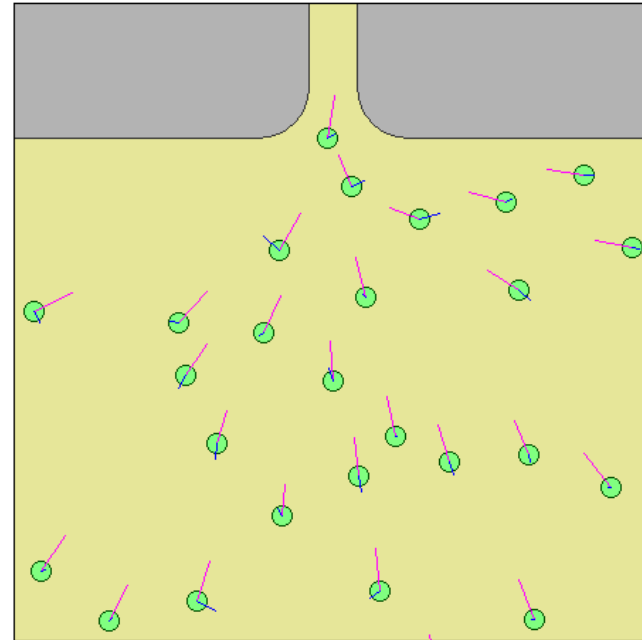
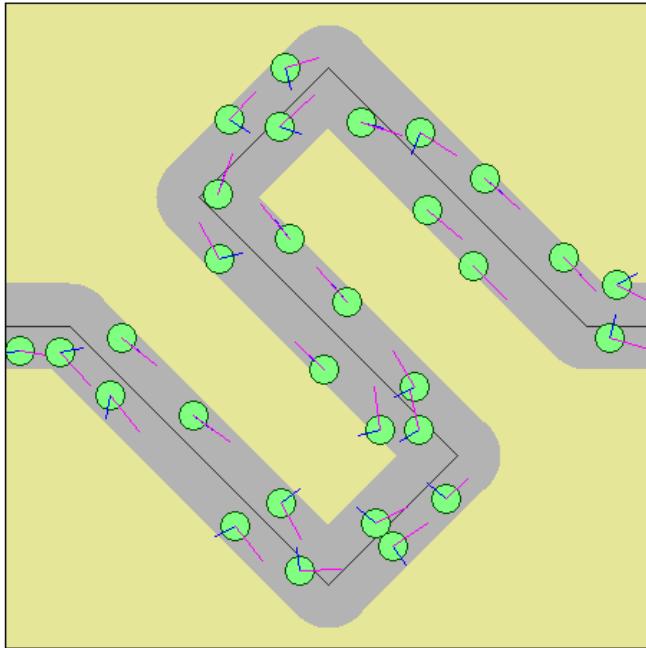
Vidéo Gish





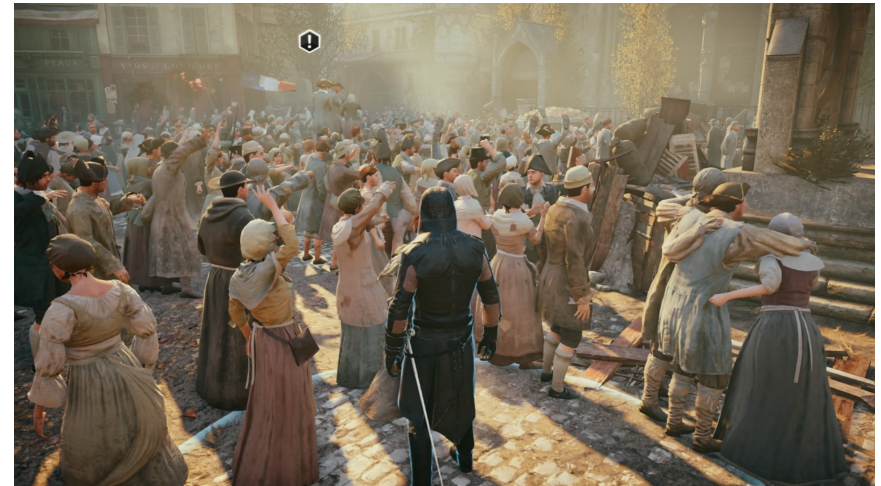
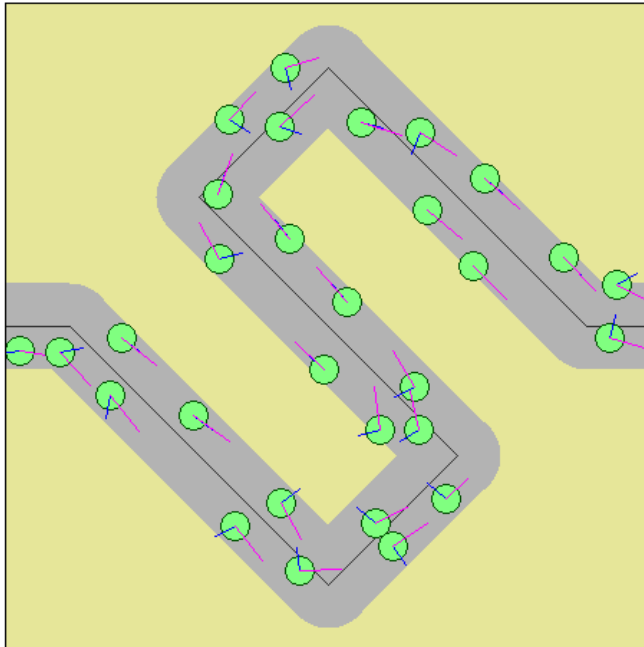
Kotsoft (youtube)





Steering behaviour

<http://www.red3d.com/cwr/steer/>



Steering behaviour

<http://www.red3d.com/cwr/steer/>

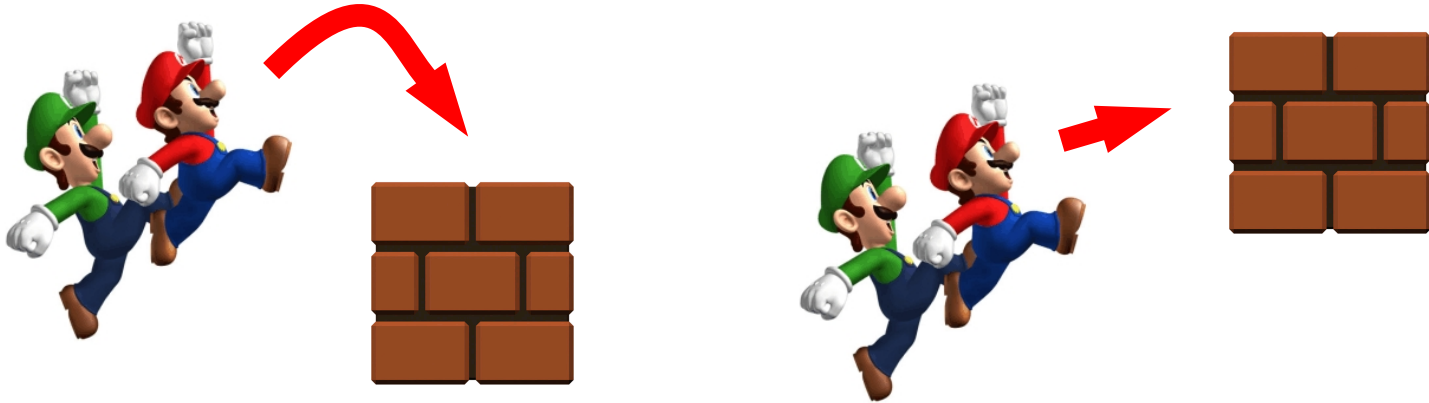
- Boucle de jeu
- Gestion du temps
- **Modèle de jeu**
 - Moteur physique
 - Gestion collisions
 - Intelligence Artificielle
- Gestion du Contrôleur
- Affichage
- Réseau

Gestion des collisions

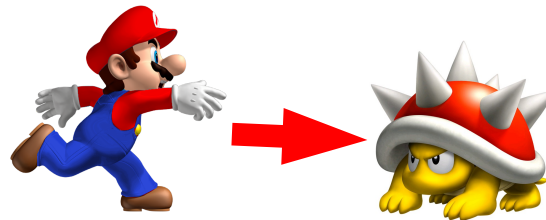


- Interactions entre objets

- Fixes



- Dynamiques



- Principes dans monde continu
 - Utilisation de « **bounding box** »



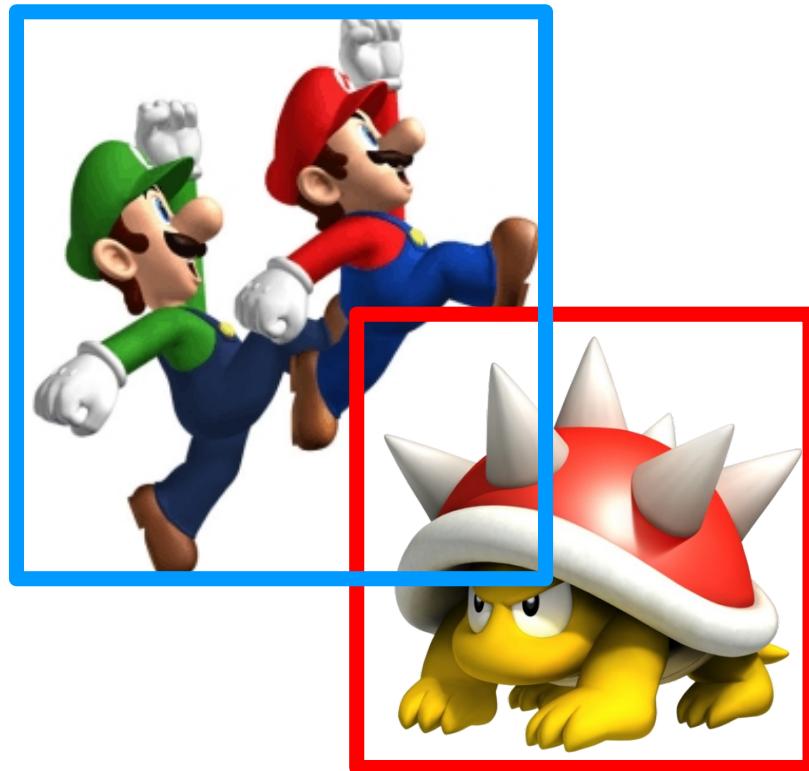
- Principes dans monde continu
 - Utilisation de « **bounding box** »



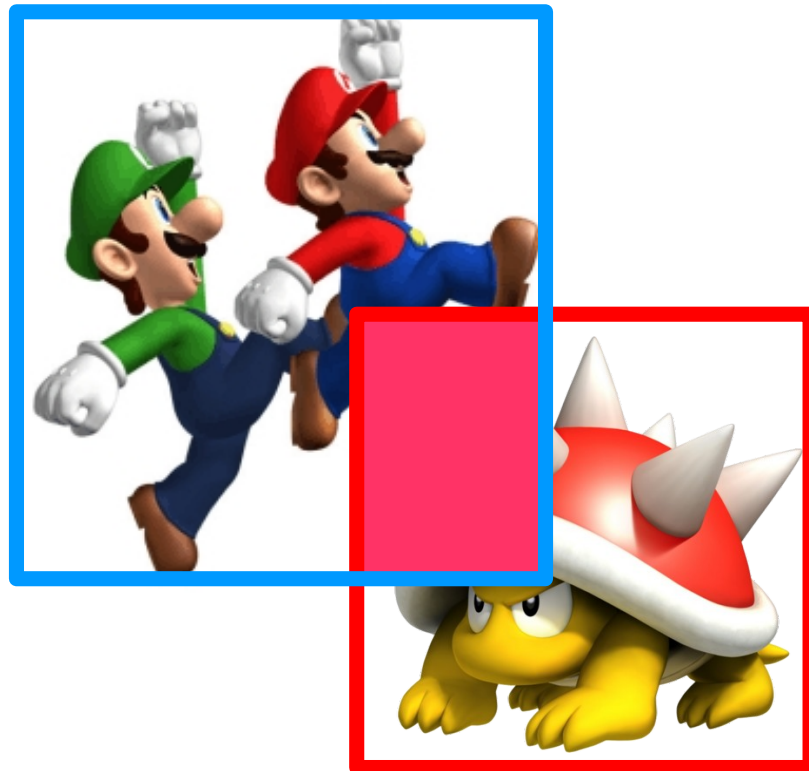
- Principes dans monde continu
 - Utilisation de « **bounding box** »



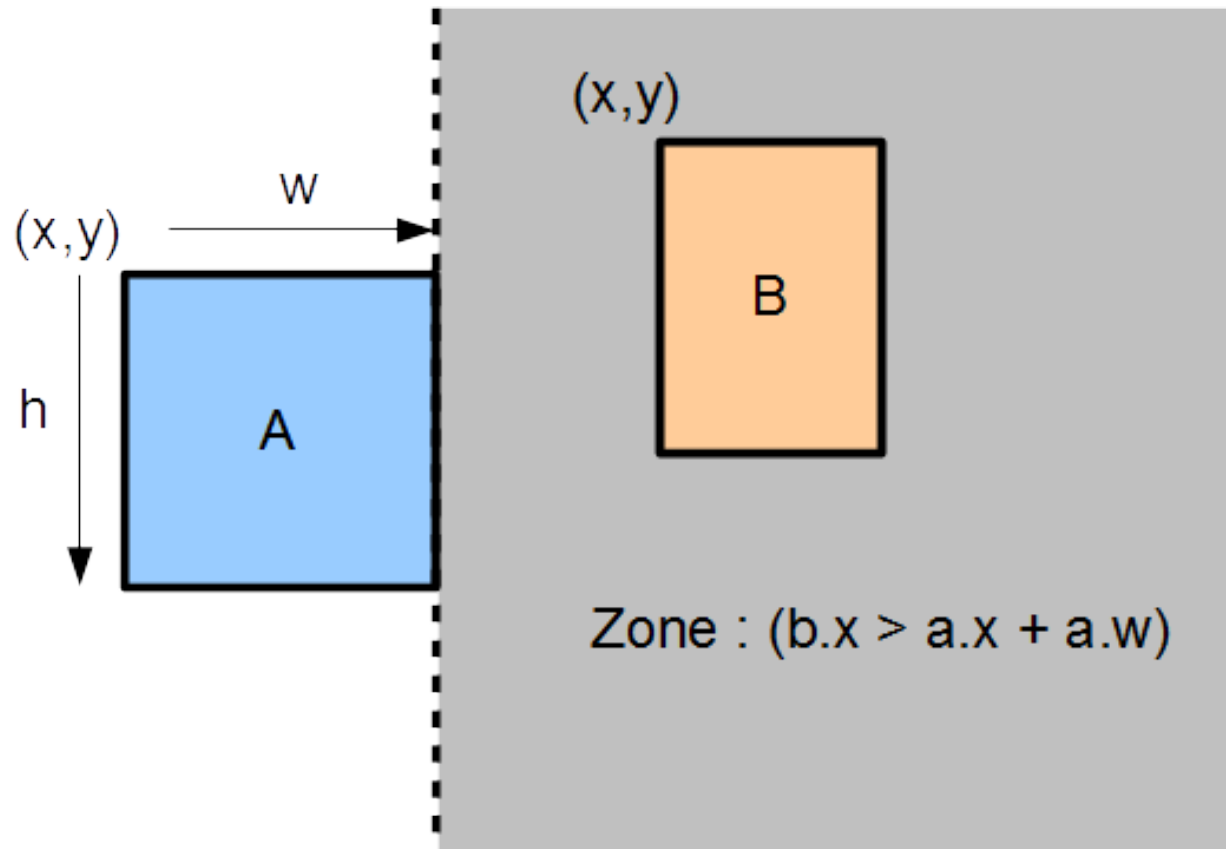
- Principes dans monde continu
 - Utilisation de « **bounding box** »



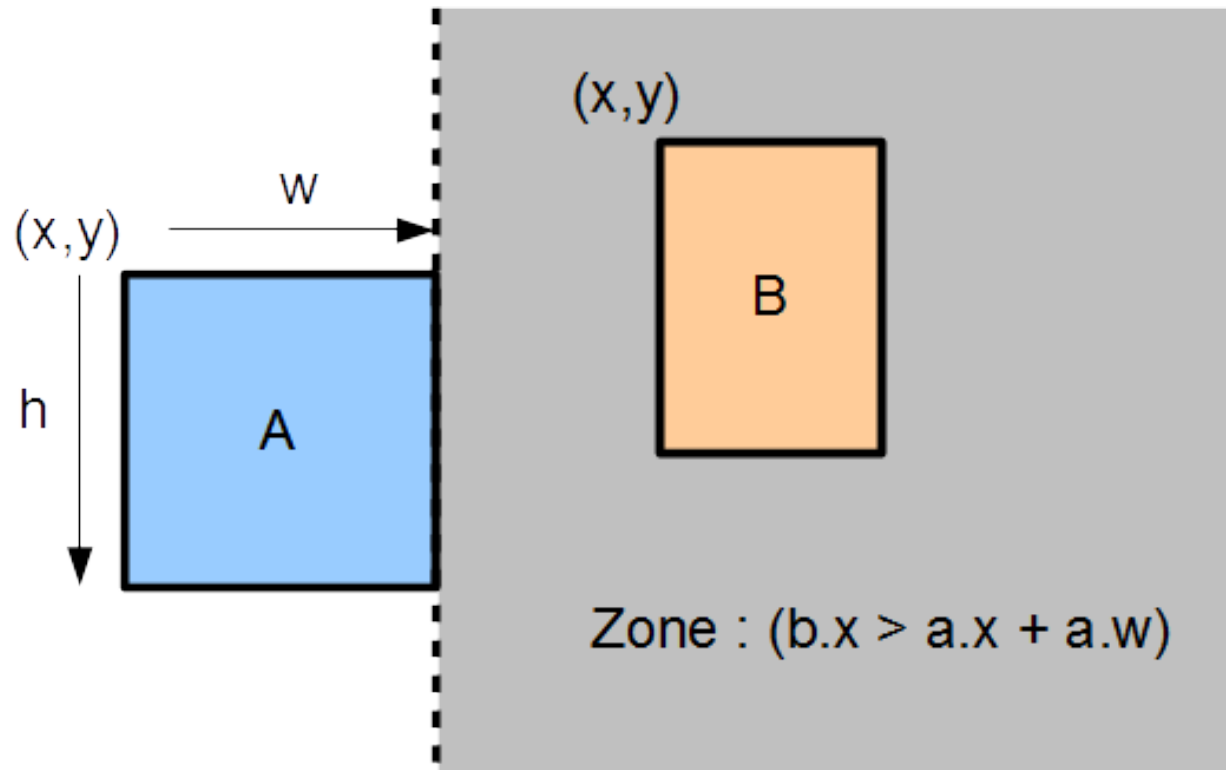
- Principes dans monde continu
 - Utilisation de « **bounding box** »



- Intersection de rectangles



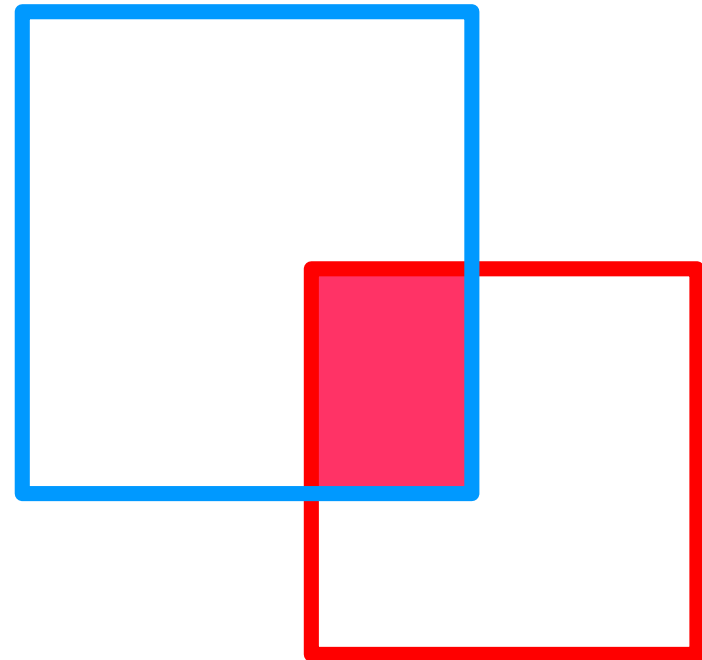
- Intersection de rectangles



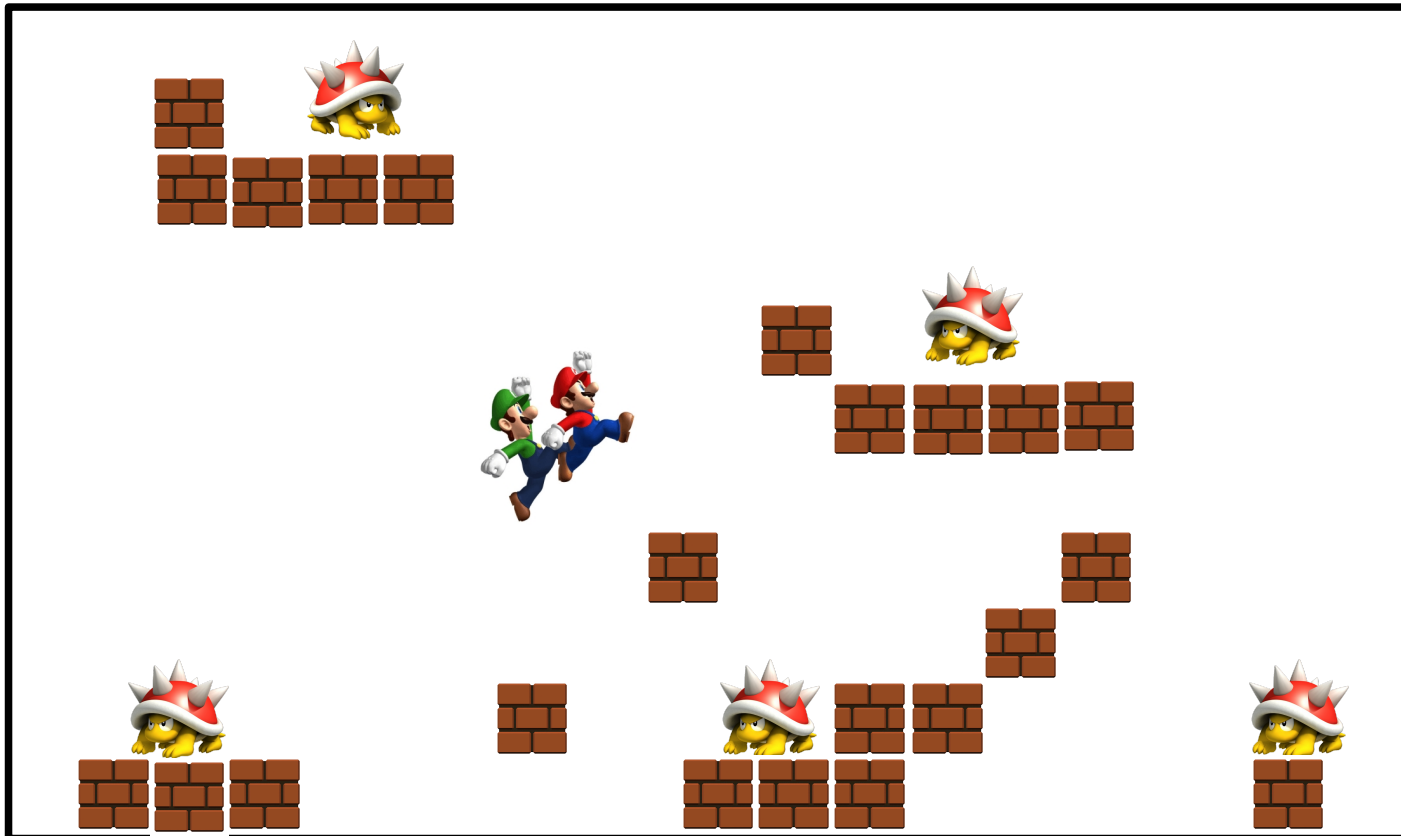
Pas intersection à droite si $(b.x > a.x + a.w)$

- Intersection de rectangles
 - teste absence de collision dans les 4 directions

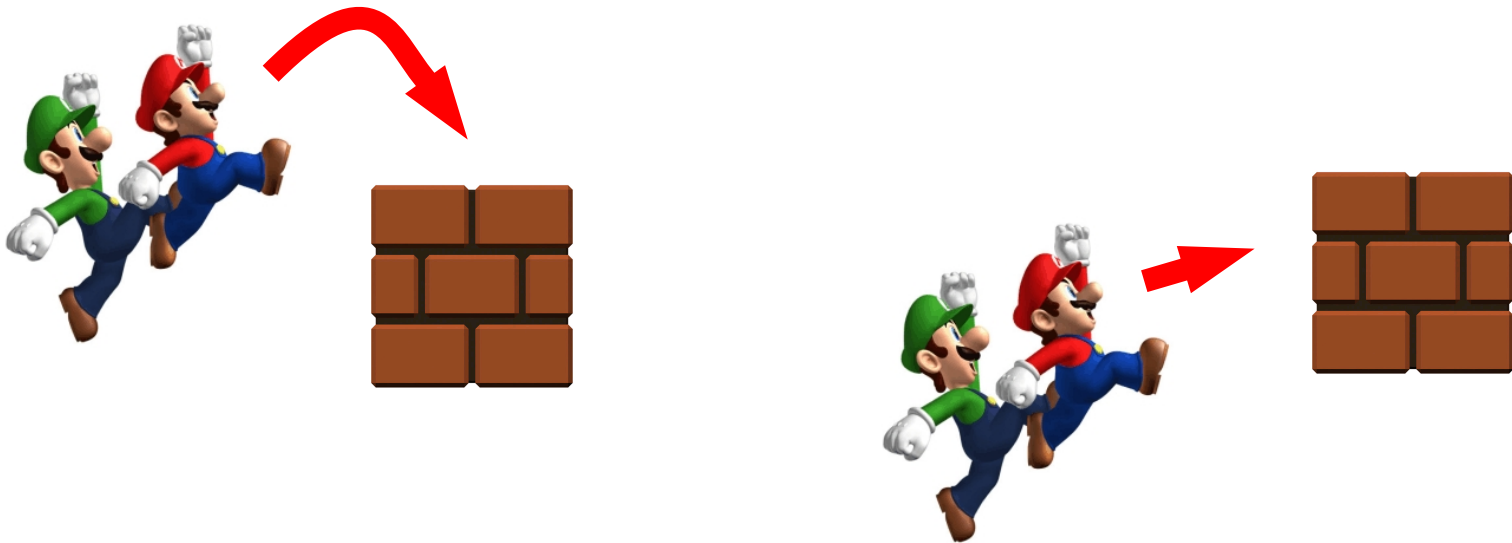
```
public boolean collide(Rect a, Rect b)
{
    If (
        (a.x >= b.x + b.w)
        || (a.x + a.w <= b.x)
        || (a.y >= b.y + b.h)
        || (a.y + a.h <= b.y))
    {
        return false;
    }
    return true;
}
```



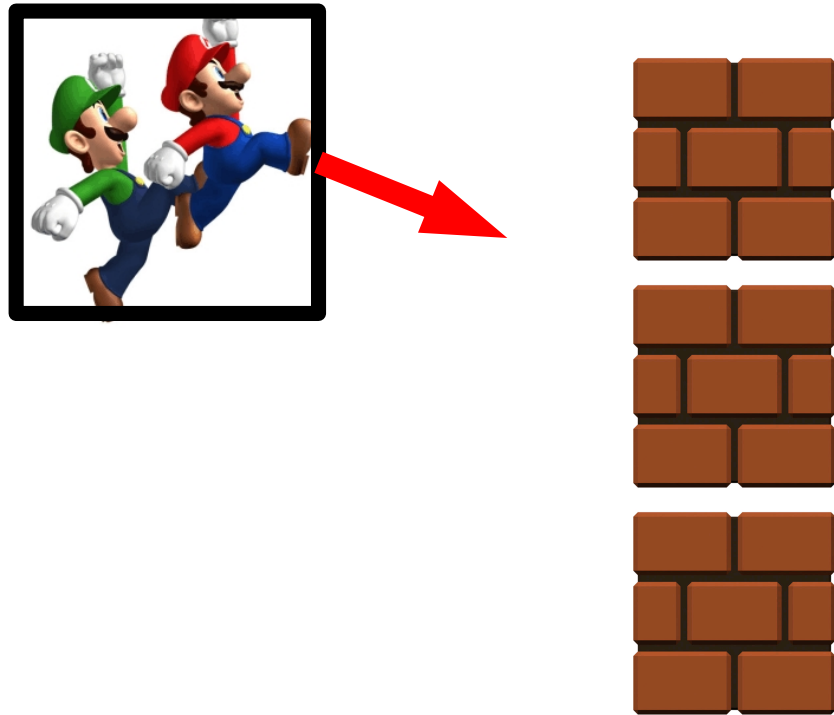
- Question ouverte
 - recherche des objets proches ?



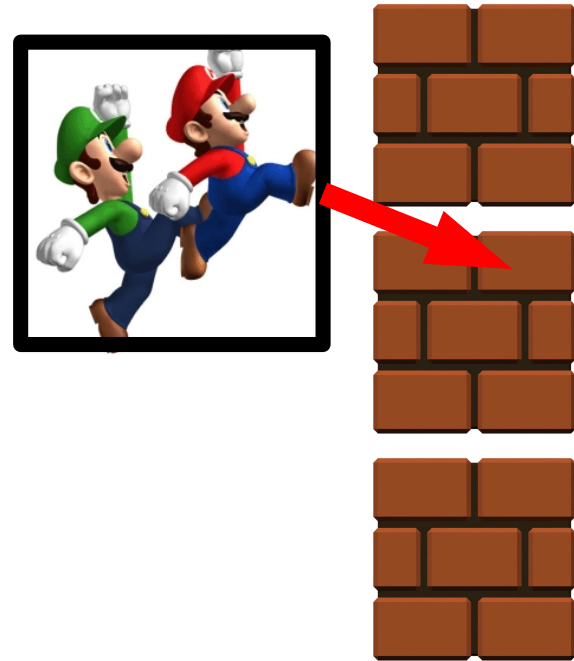
- **Prise en compte du résultat**
 - Dépend de l'angle



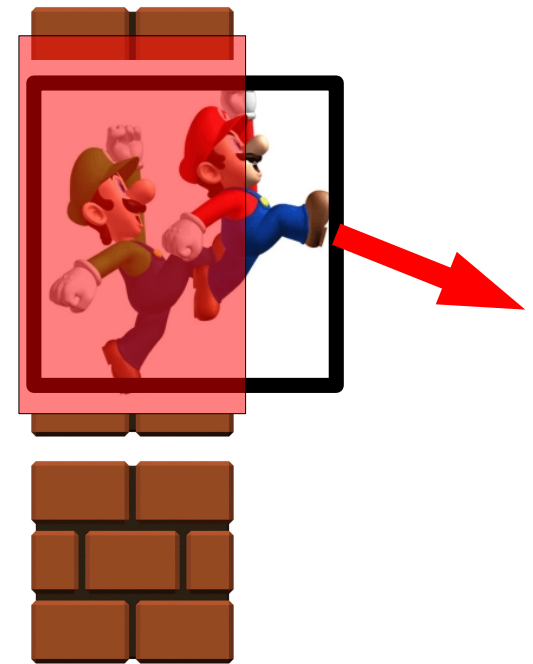
- Prise en compte du résultat



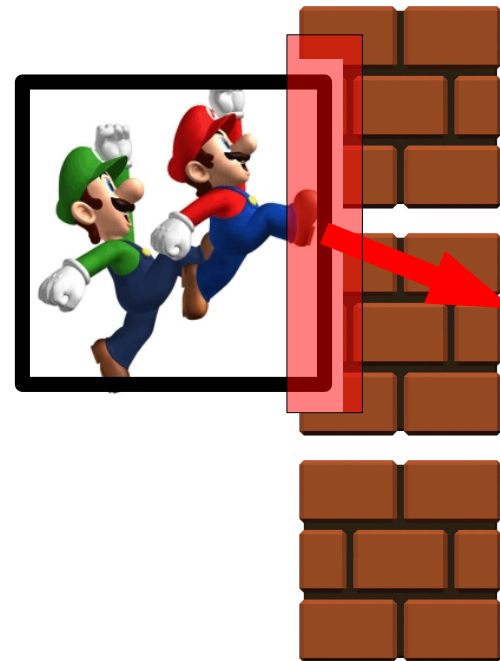
- Prise en compte du résultat



- **Prise en compte du résultat**
 - Cas problématique qui ne devrait pas arriver



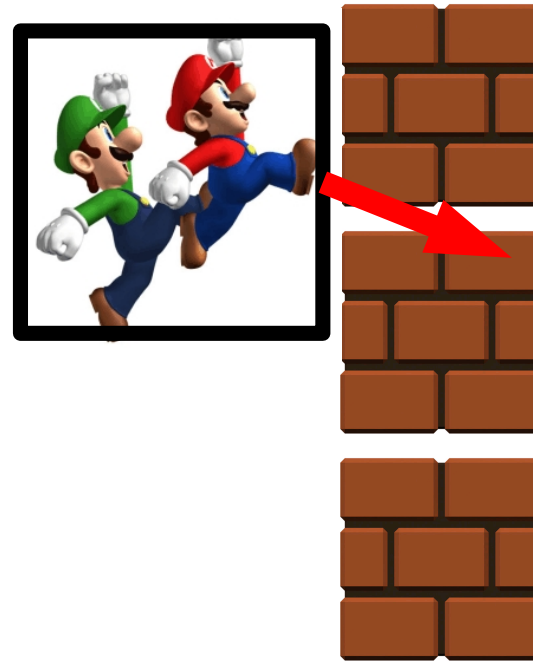
- **Prise en compte du résultat**
 - Zone intersection est petite (dt petit)



- **Prise en compte du résultat**
 - Zone intersection est petite (dt petit)

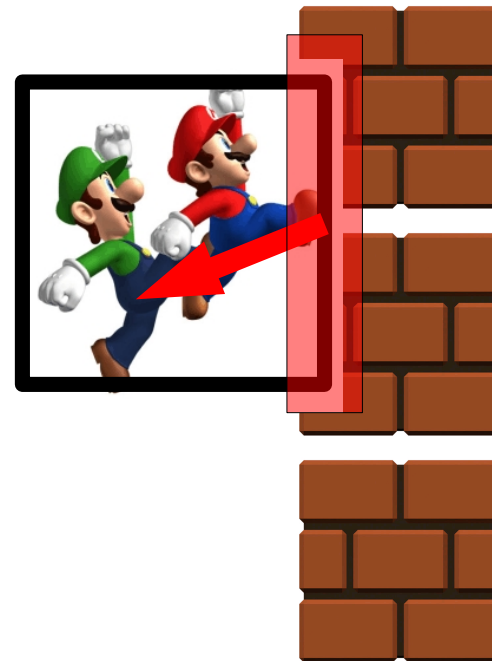
- **Solution**

1. Revenir passé

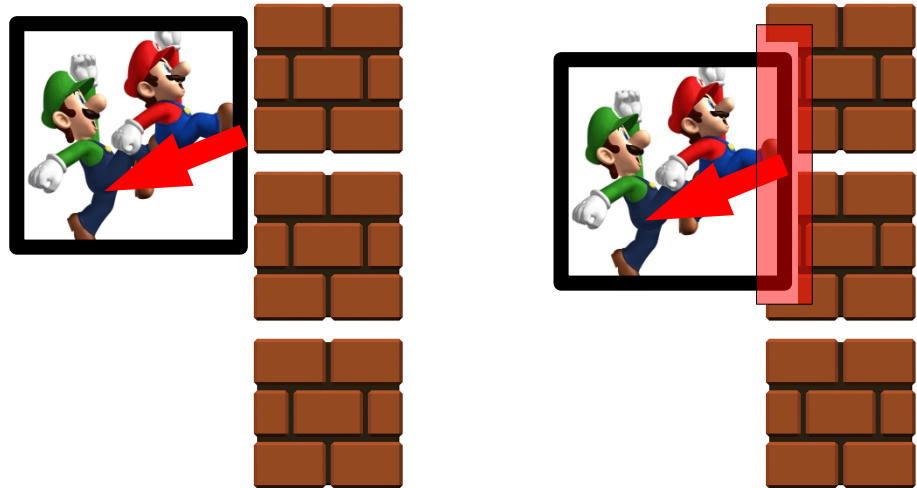


- **Prise en compte du résultat**
 - Zone intersection est petite (dt petit)

- **Solution**
 1. Revenir passé
 2. Approximation



- **Prise en compte du résultat**
 - Zone intersection est petite (dt petit)
- **Solution**
 1. Revenir passé
 2. Approximation



- **Solution**
 - Revenir passé

```
// permet de retourner la direction de collision
public static int direction(Objet o, Objet o2) {

    // on regarde simplement les anciennes positions
    double oldx = o.px - o.vx;
    double oldy = o.py - o.vy;

    double oldx2 = o2.px - o2.vx;
    double oldy2 = o2.py - o2.vy;

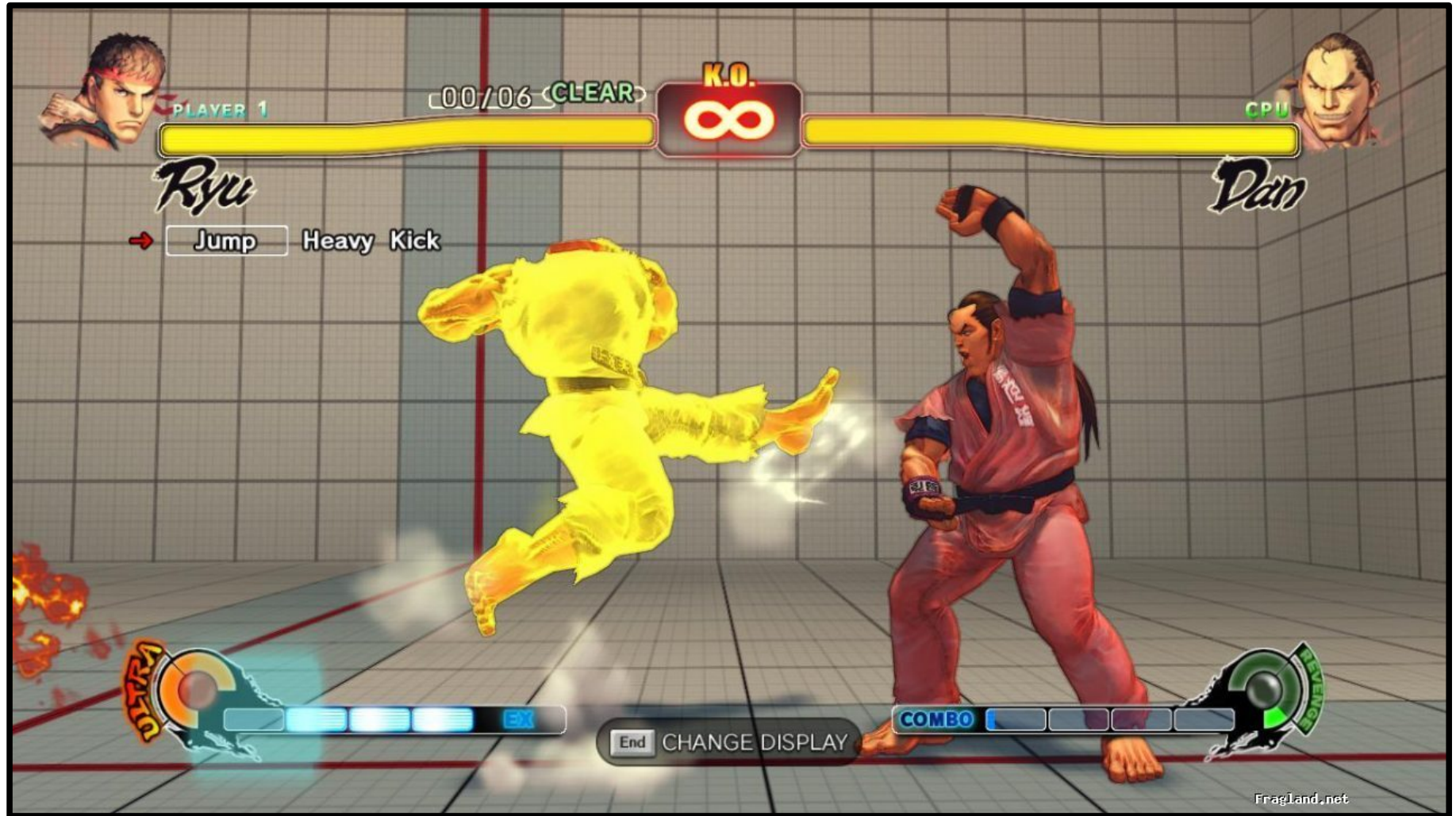
    // reprend chacune des conditions
    if (oldx >= oldx2 + o2.width) {
        // o vient de la droite de o2
        return (DROITE);
    }
    if (oldx + o.width <= oldx2) {
        // o vient de la gauche de o2
        return (GAUCHE);
    }
    if (oldy >= oldy2 + o2.height) {
        // o vient du haut
        return (HAUT);
    }
    if (oldy + o.height <= oldy2) {
        // o vient du bas
        return (BAS);
    }
    throw new AssertionError("Collision bizarre");
}
```

```
// test de collision pour chaque mur
for (Objet obj : m.objets) {
    if (Collision.collusion(m.balle, obj)) {
        m.balle.collusion = true;
        int direction = Collision.direction(m.balle, obj);
        // revient avant collision
        m.balle.px = m.balle.px - m.balle.vx;
        m.balle.py = m.balle.py - m.balle.vy;

        // modifie la vitesse
        if ((direction == Collision.GAUCHE)
            || (direction == Collision.DROITE))
            m.balle.vx = -m.balle.vx;
        else
            m.balle.vy = -m.balle.vy;
    } else
        m.balle.collusion = false;
}
```

- **Gestionnaire de collision**
 - Élément de base dans jeu
 - Monde continu
- **Génère les événements**
 - Rencontre
 - Obstacles
- **Présent dans la plupart des jeux**

Jeux de combat 2D(.5)



- Hitbox / Hurtbox /Blockbox (Street Fighter 4)

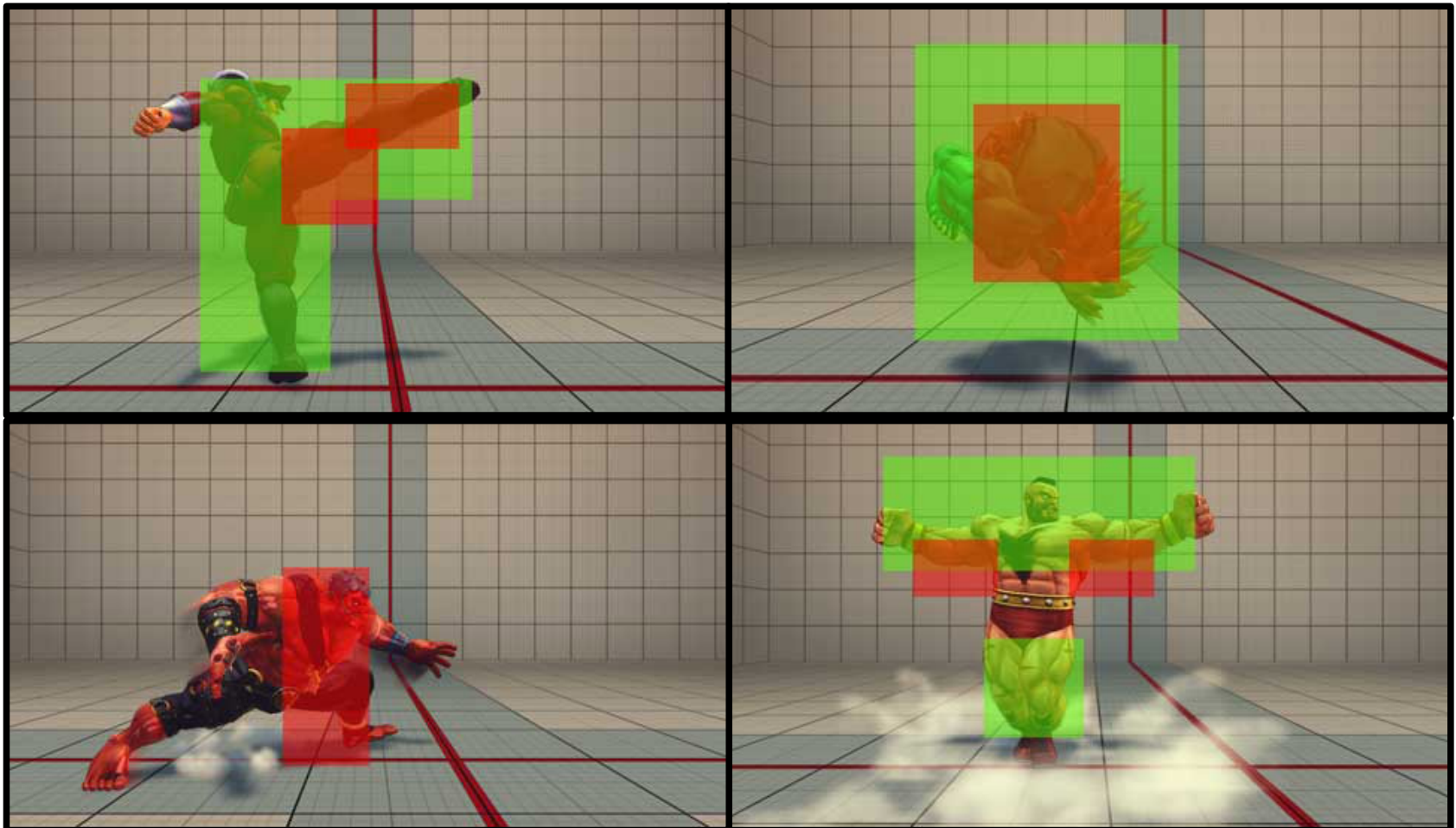


Diagramme de classe

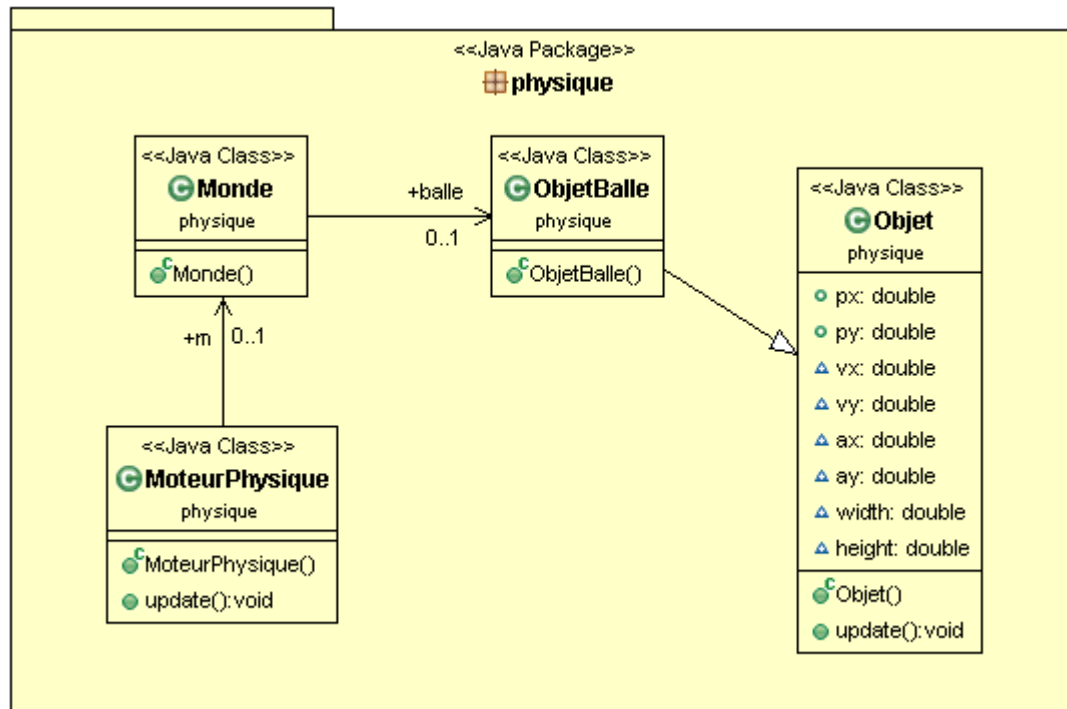


Diagramme de classe

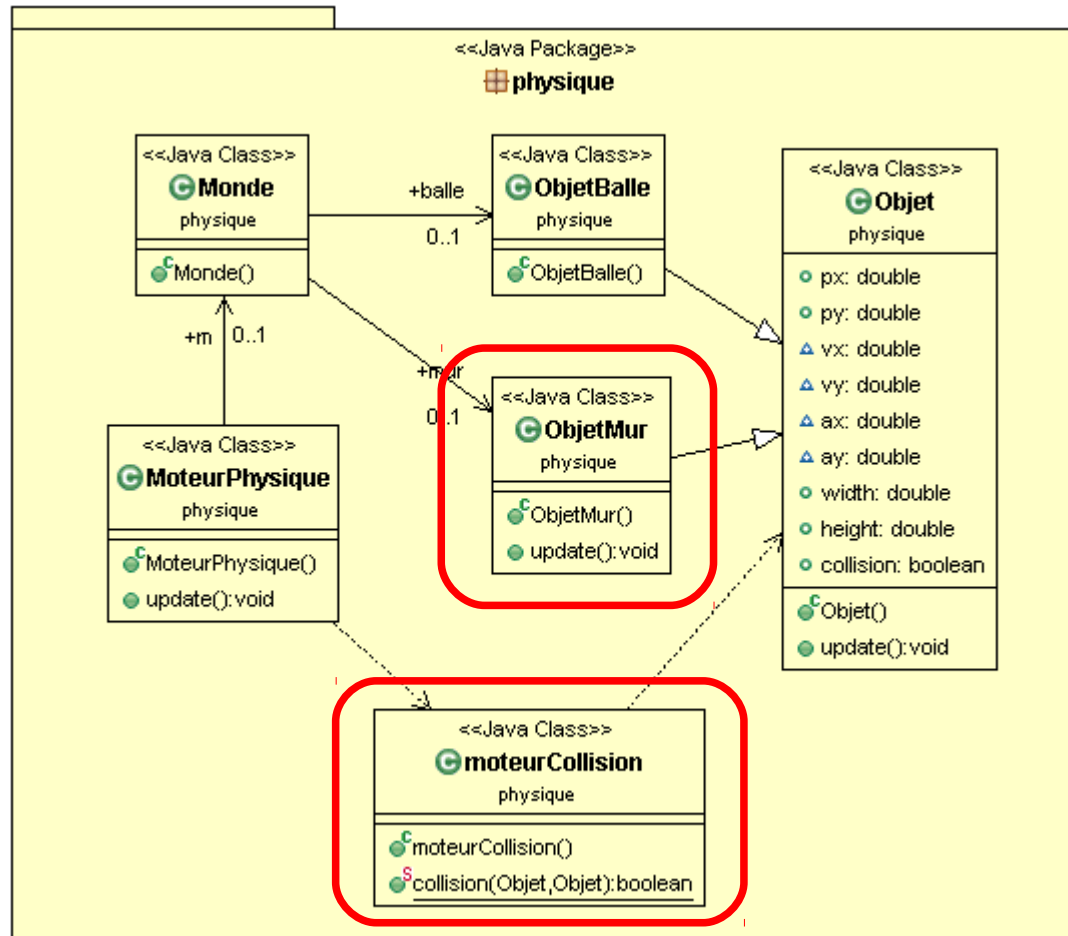
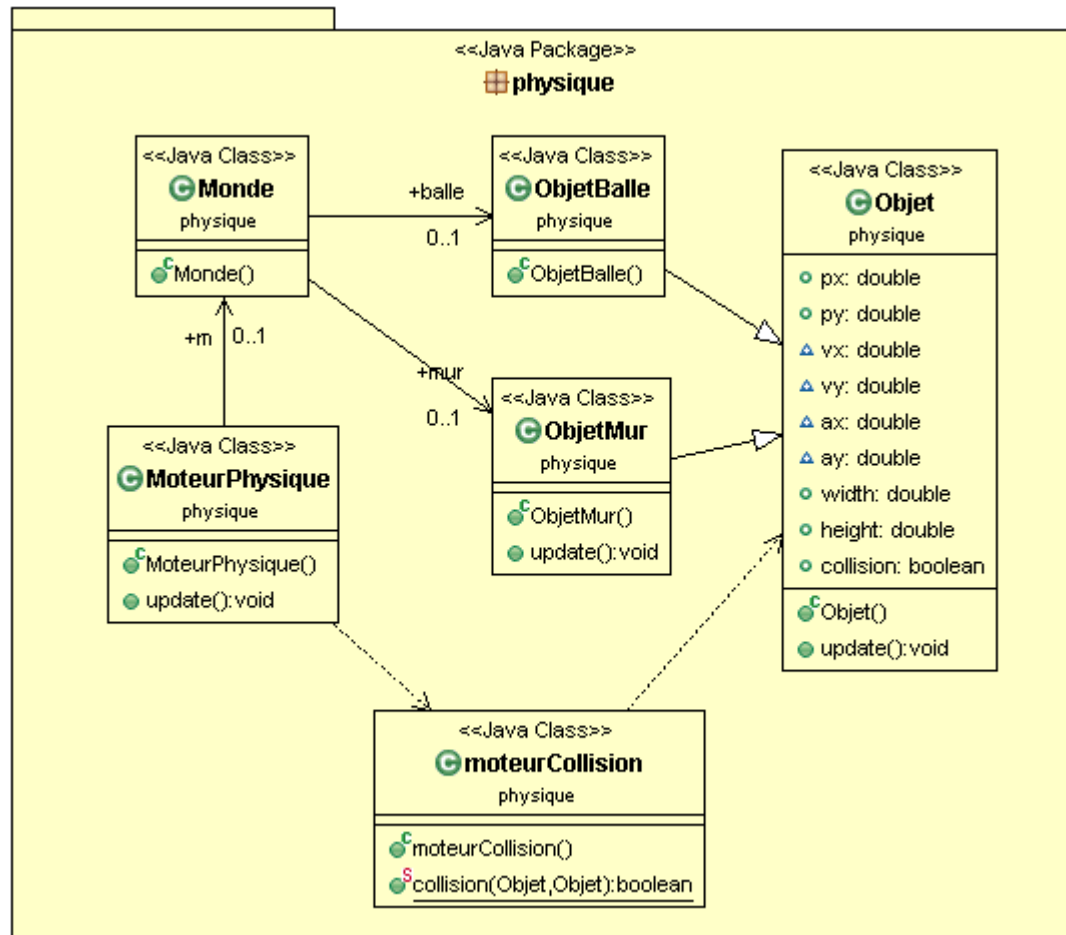


Diagramme de classe



Démonstration Partie 4

Gestion des collisions

04x01 – couleur quand collision

04x02 – rebond quand collision

04x03 – collision avec plusieurs objets

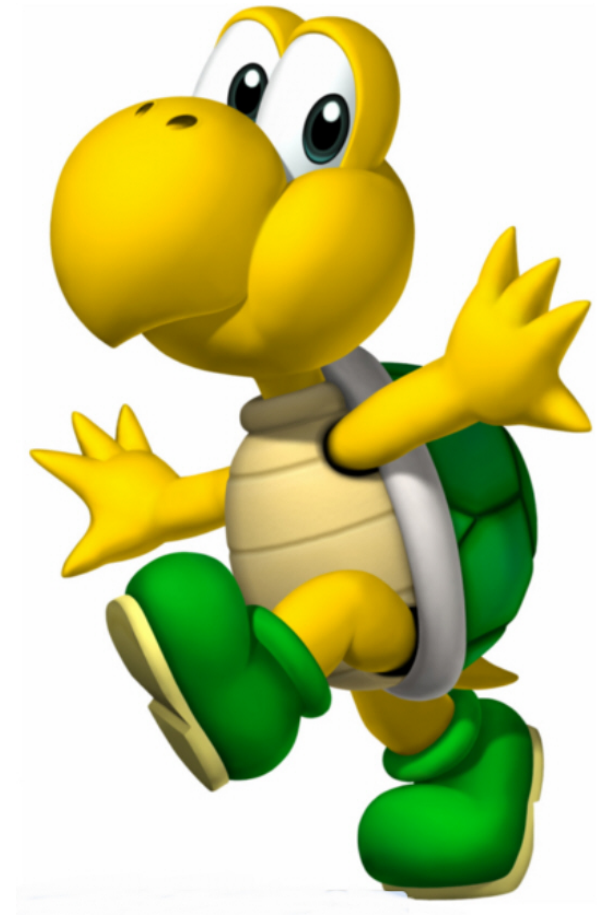
04x04 – collision sol = objet

04x05 – collision et objets dynamiques

04x06 – changement taille (modele suit)

- Boucle de jeu
- Gestion du temps
- **Modèle de jeu**
 - Moteur physique
 - Gestion collisions
 - Intelligence Artificielle
- Gestion du Contrôleur
- Affichage
- Réseau

- **Gérer un/des ennemis**
 - Plus (ou moins) intelligents
- **Comportement**
 - chaque itération, décision ?
- **Dans l'update**
 - Pour chaque ennemi



- **Plusieurs approches**
 - Cf transparents Intelligence artificielle
- **Un moyen**
 - Machine état fini
- **Définir**
 - Comportements
 - Actions associées
 - Changement de comportement

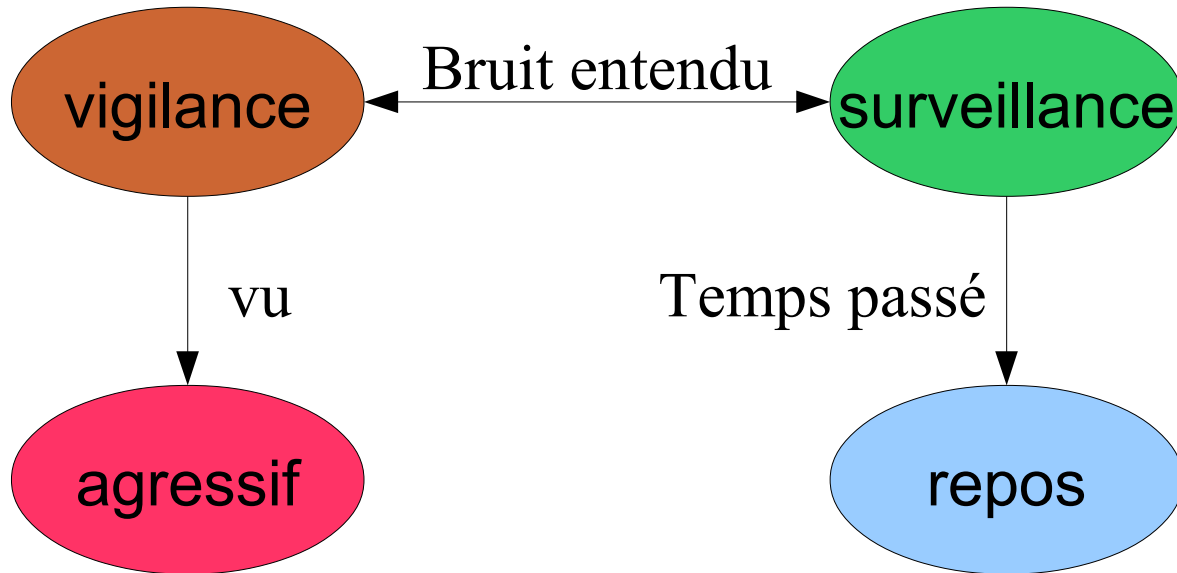




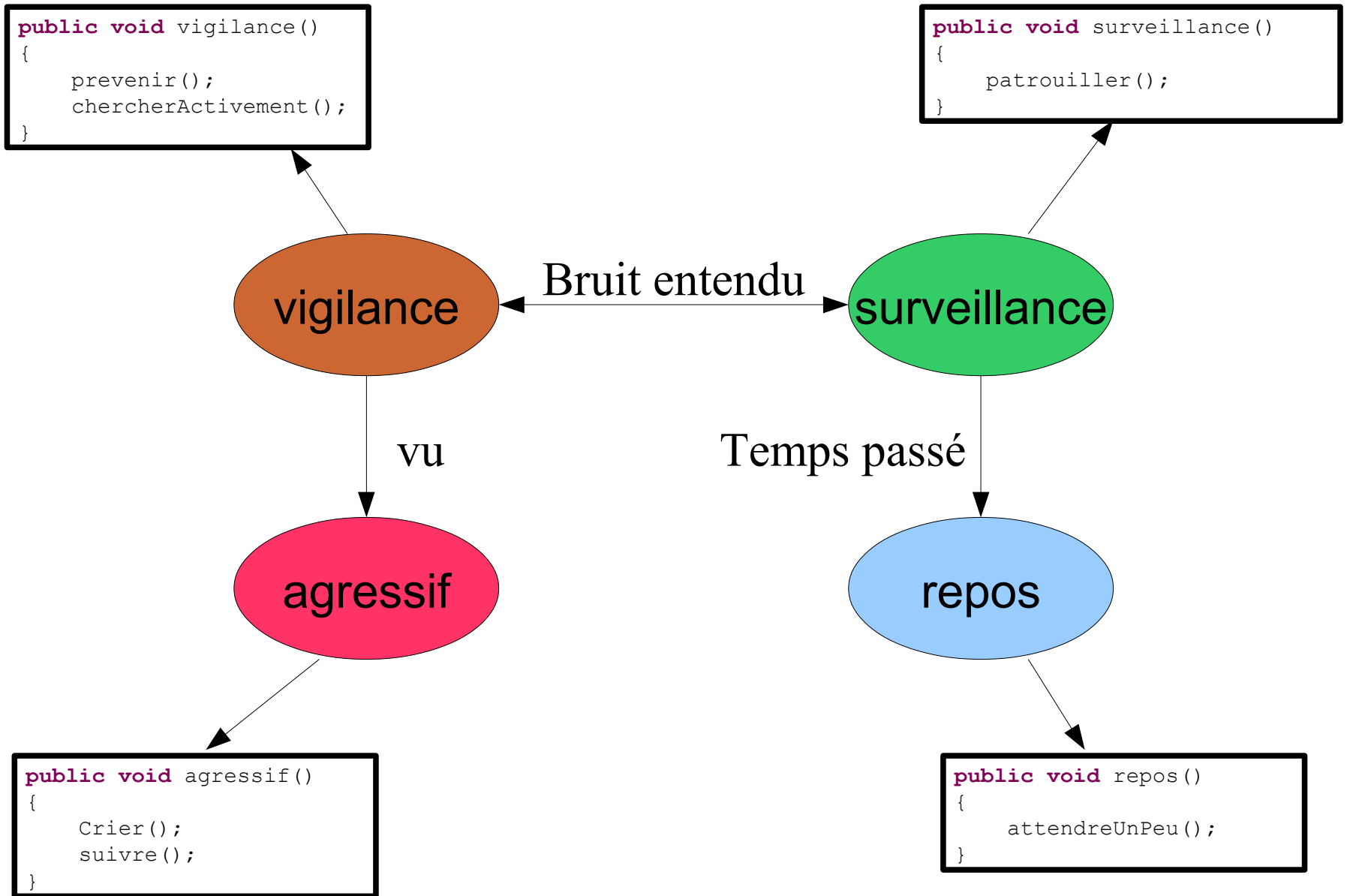
Exemple « beyond good and evil »



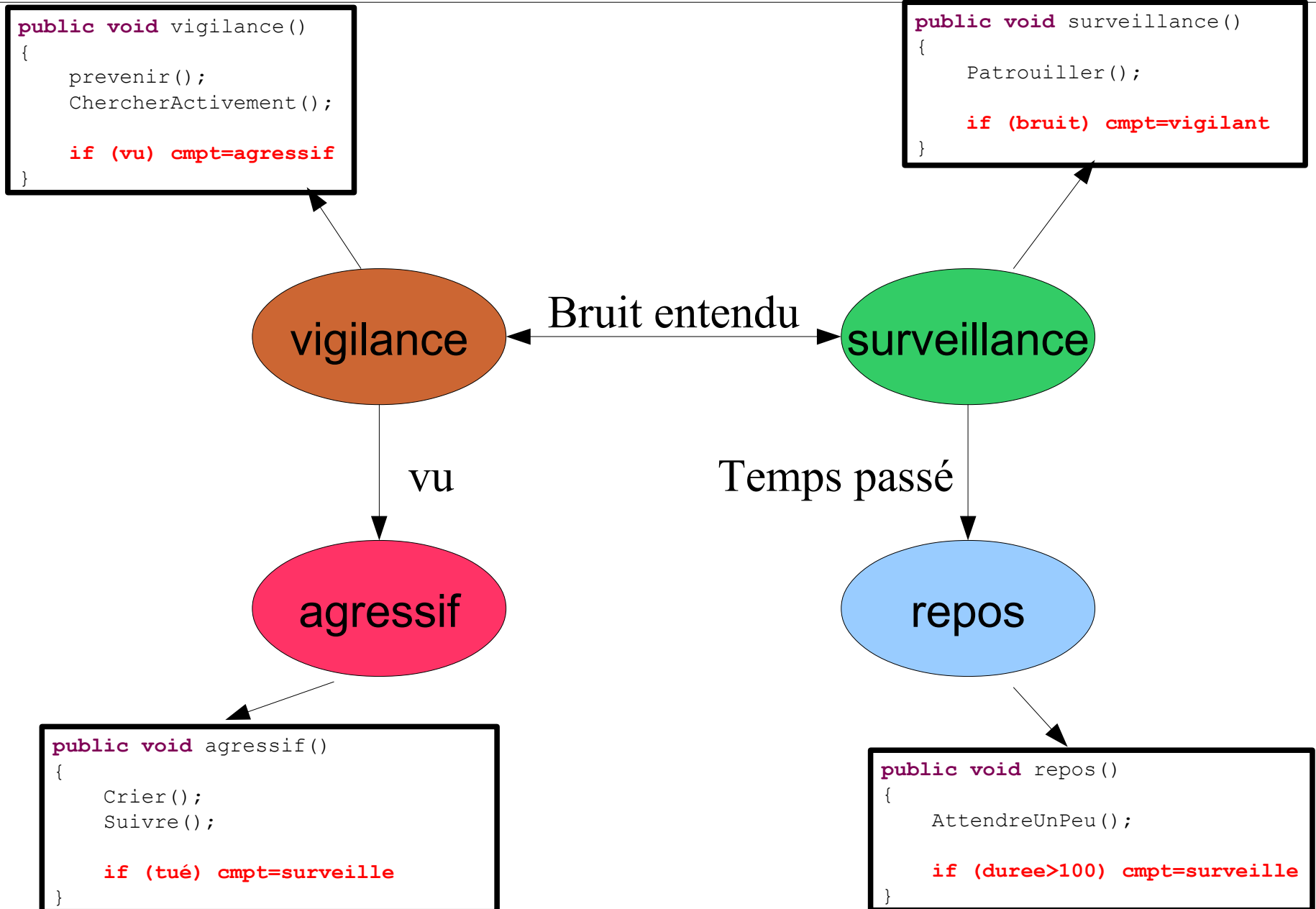
Exemple « beyond good and evil »



Exemple « beyond good and evil »



Exemple « beyond good and evil »

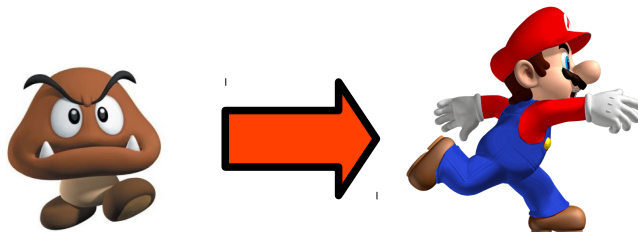


Un exemple

- Deux comportements
 - Promenade



- Attaque



- Deux comportements

- Promenade



"Promene"

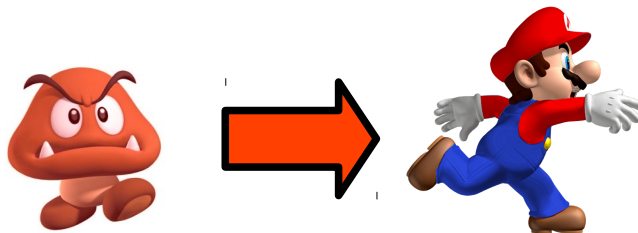
Action

→ Avancer jusque obstacle

Transition

→ Mario proche => attaque

- Attaque



"Attaque"

Action

→ Suivre Mario

→ Monter obstacles

Transition

→ Mario trop loin => promène

- Décision == aiguillage

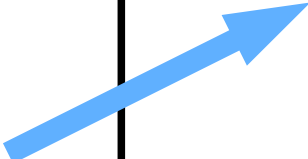
```
public void action() {
    switch (this.comportement) {

        //s'il se promene
        case "Promene":
            this.promene();
            //change de comportement ?
            if (this.proche(mario))
                comportement = "Attaque";
            break;


        //s'il attaque
        case "Attaque":
            this.attaque();
            //change de comportement ?
            if (!this.proche(mario))
                comportement = "Promene";
            break;
    }
}
```


- Décision == aiguillage

```
public void action() {  
    switch (this.comportement) {  
  
        //s'il se promene  
        case "Promene":  
            this.promene();  
            //change de comportement ?  
            if (this.proche(mario))  
                comportement = "Attaque";  
            break;  
  
        //s'il attaque  
        case "Attaque":  
            this.attaque();  
            //change de comportement ?  
            if (!this.proche(mario))  
                comportement = "Promene";  
            break;  
    }  
}
```



```
public void promene() {  
    //essaie d'avancer  
    avance(direction);  
  
    //si un obstacle se retourne  
    if (obstacle)  
        changeDirection();  
}
```



```
public void attaque() {  
    //se tourne dans la bonne direction  
    if (direction != this.vers(Mario))  
        changeDirection();  
  
    //avance vers mario  
    avance(direction);  
}
```

- **Machine états finis**
 - Pas une solution dans l'absolu
- **Moyen d'organiser comportements**
 - Hiérarchie
 - Succession
- **IA beaucoup d'autre choses**
 - Planifier, apprendre, s'adapter

Diagramme de classe

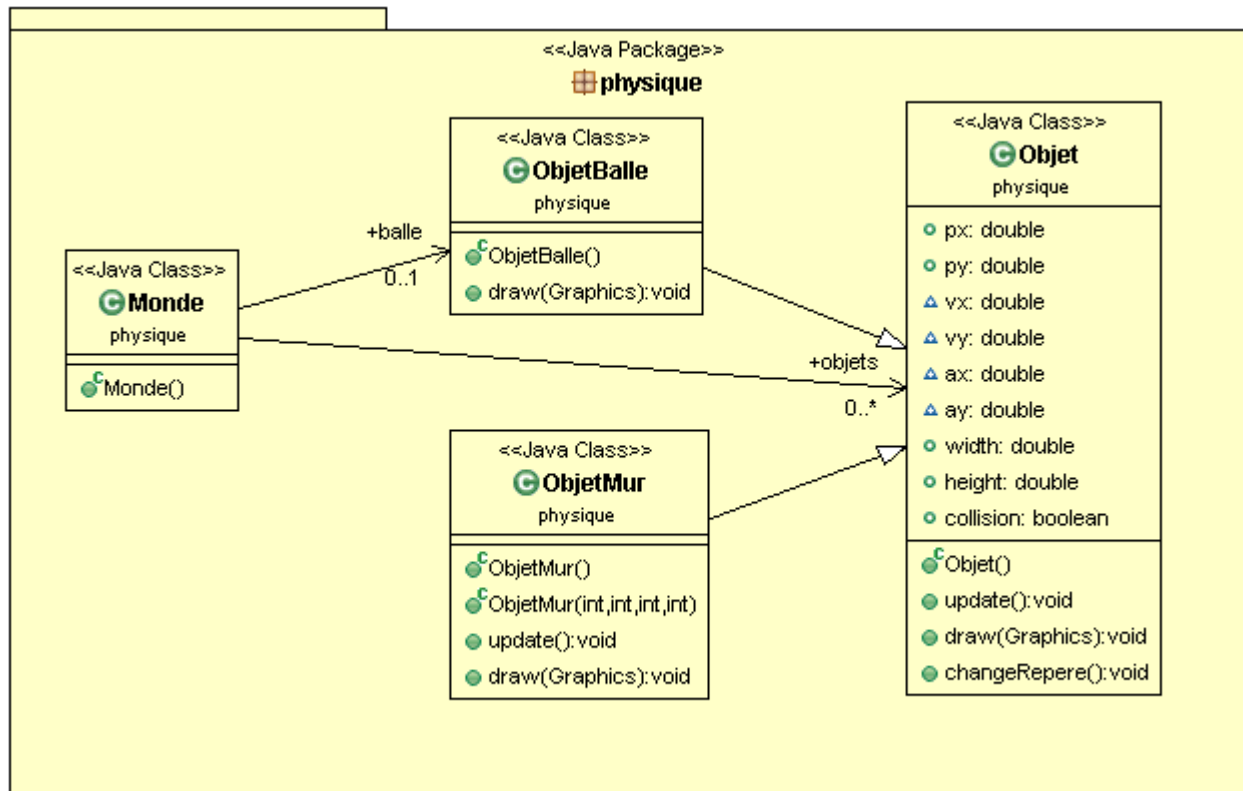
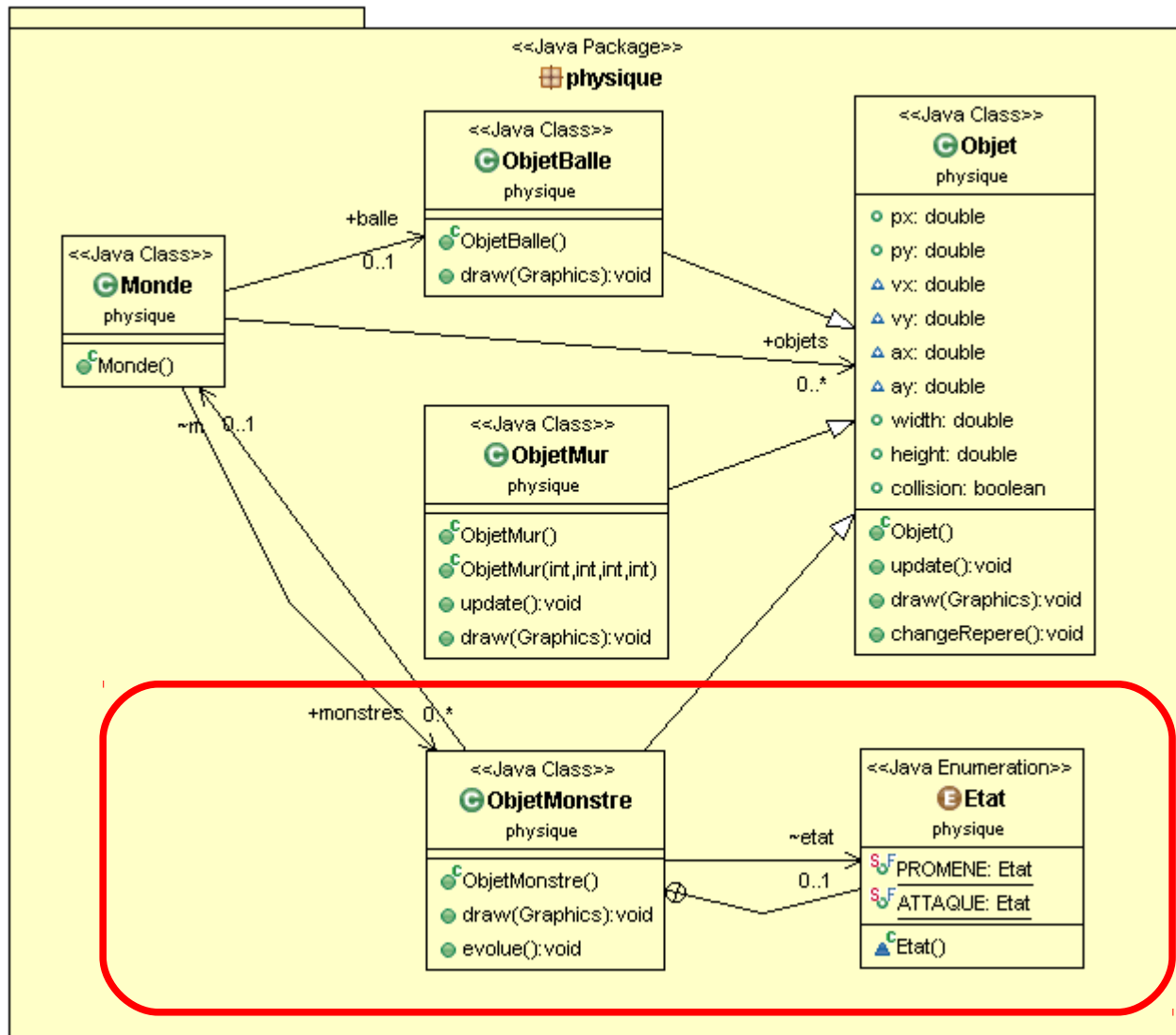


Diagramme de classe



Démonstration Partie 5

Intelligence artificielle

05x01 – IA simple

05x02 – IA avec rebond contre murs

05x03 – IA avec machine à états

05x04 – plusieurs monstres

- **Lois du monde**
 - lois physiques
 - Règles du jeu
- **Gestion des collisions**
 - Bounding box
- **Intelligence artificielle**
 - Machine états finis

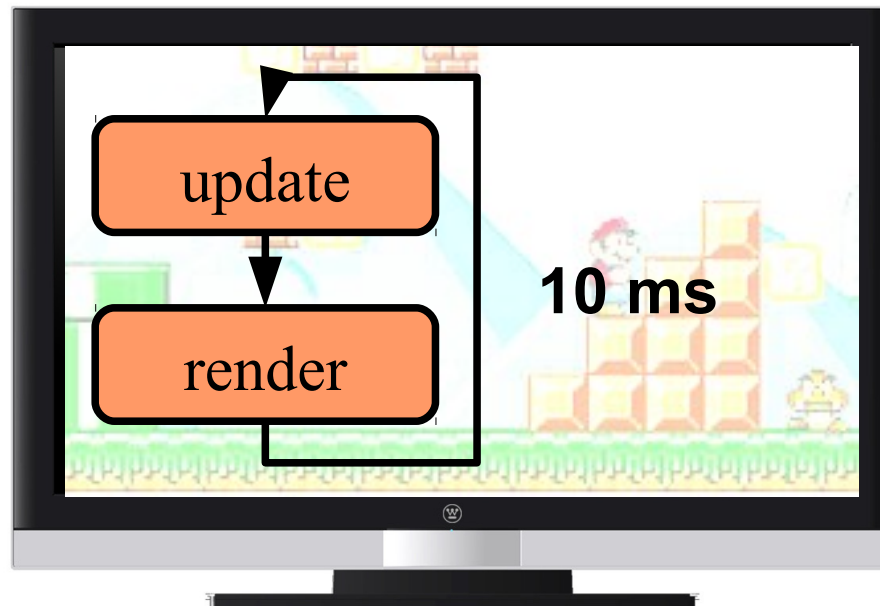
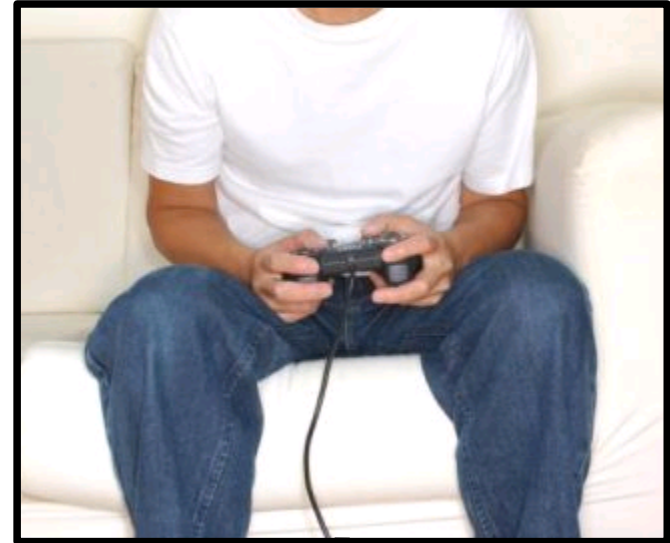
- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
- Réseau

- Ne pas rater un événement
 - Mauvaise jouabilité
- Mise à jour rapide
 - Bloquant pour l'écoute
- Centraliser dans l'update
 - Pouvoir gérer facilement les interactions

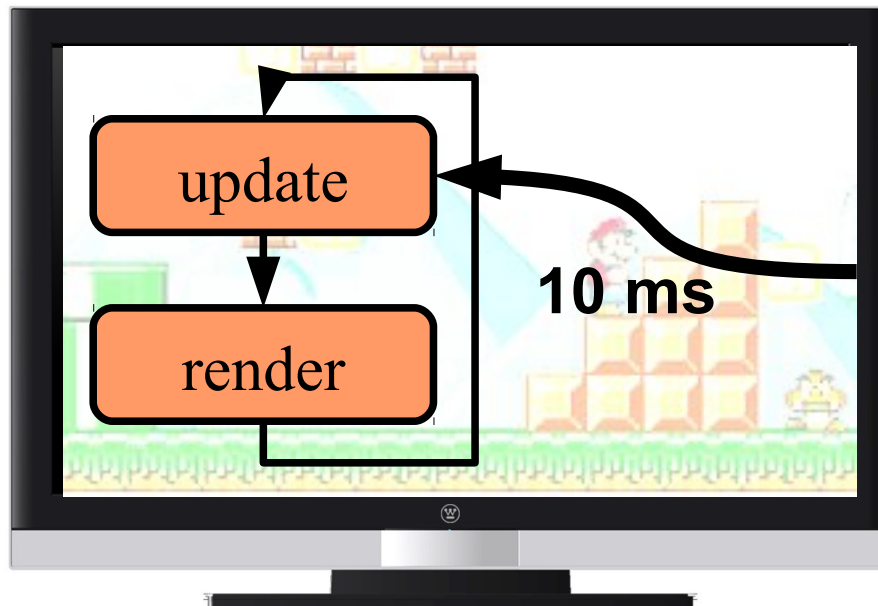
- Actions du joueur



- Actions du joueur

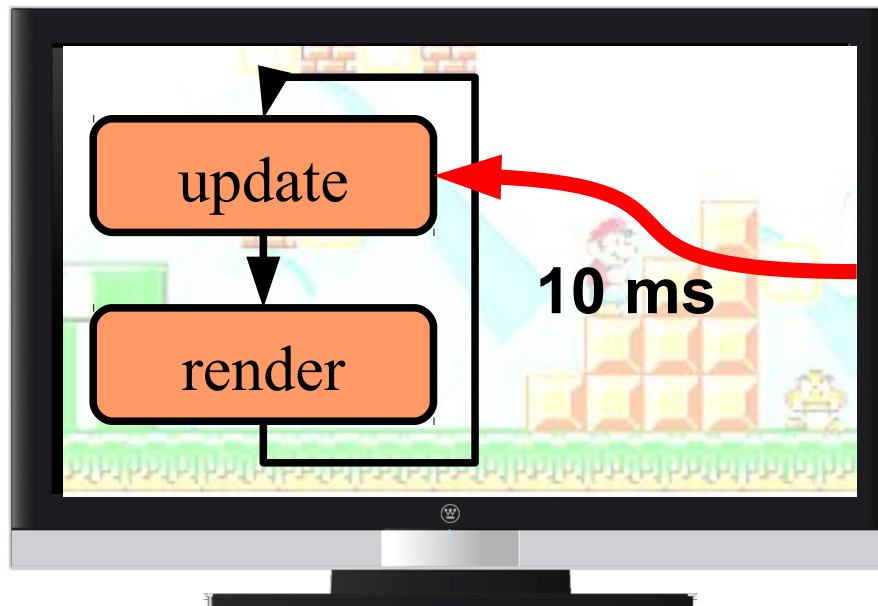


- Actions du joueur



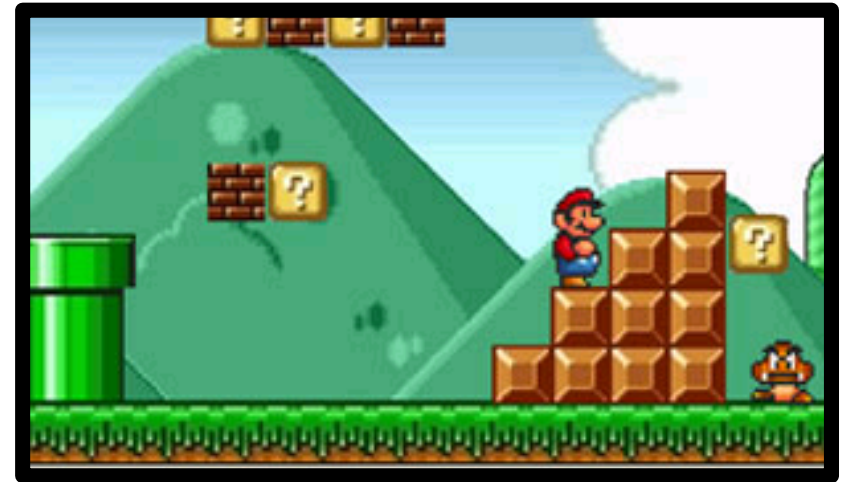
- Actions du joueur

Or action est asynchrone



- Tout centraliser pour gérer les conflits
- Utilisateur == extérieur
 - Programmation événementielle Asynchrone

```
public void keyPressed(KeyEvent e) {  
    if (e.getKeyCode() == e.VK_LEFT) {  
        mario.x-- ;  
    }  
    if (e.getKeyCode() == e.VK_RIGHT) {  
        mario.x-- ;  
    }  
}
```



```
public void keyPressed(KeyEvent e) {  
    if (e.getKeyCode() == e.VK_LEFT) {  
        mario.x-- ;  
    }  
    if (e.getKeyCode() == e.VK_RIGHT) {  
        mario.x-- ;  
    }  
}
```





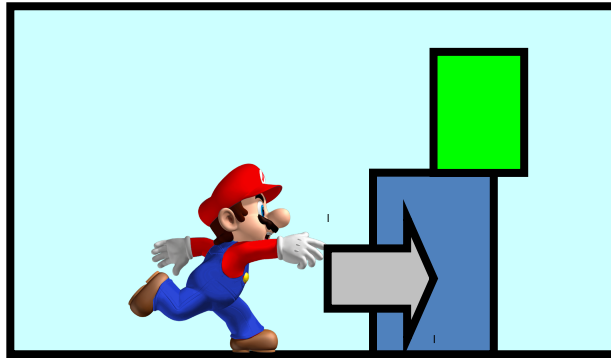
```
public void keyPressed(KeyEvent e) {  
    if (e.getKeyCode() == e.VK_LEFT) {  
        mario.x-- ;  
    }  
    if (e.getKeyCode() == e.VK_RIGHT) {  
        mario.x-- ;  
    }  
}
```



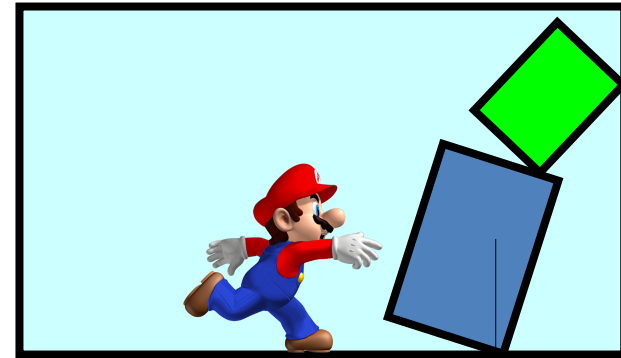
Problème concurrence d'accès

```
pub  
if  
    }  
    }  
if (e.getKeyCode() == e.VK_RIGHT) {  
    mario.x-- ;  
}  
}
```





Géré par clavier

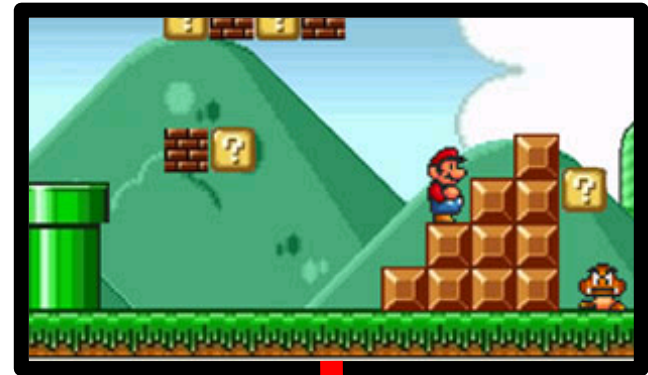


Géré par update

- **Interaction avec le monde**
 - Ne pas déplacer un personnage bloqué
 - Modifie les comportements du monde
 - Génère des déplacements

- **Approche**
 - Resynchroniser par une classe intermédiaire
- **Classe contrôle**
 - Booléen gauche, droite, ...
- **Toute modification dans update**
 - synchroniser

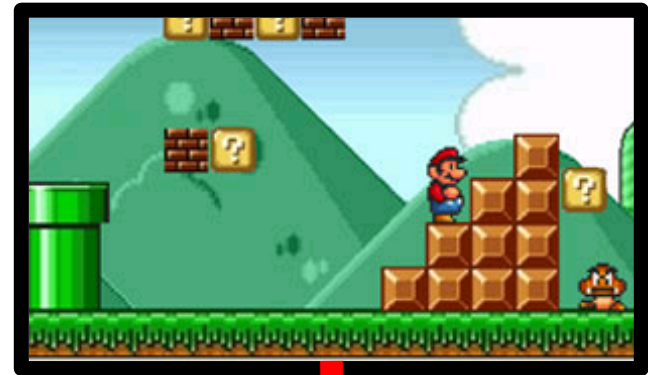
Principe



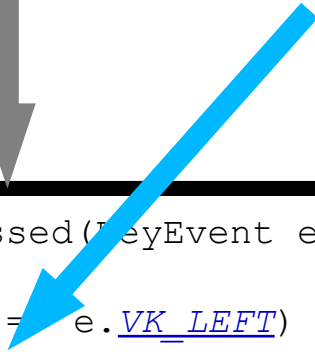
```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            Mario.x++ ;  
        }  
}
```



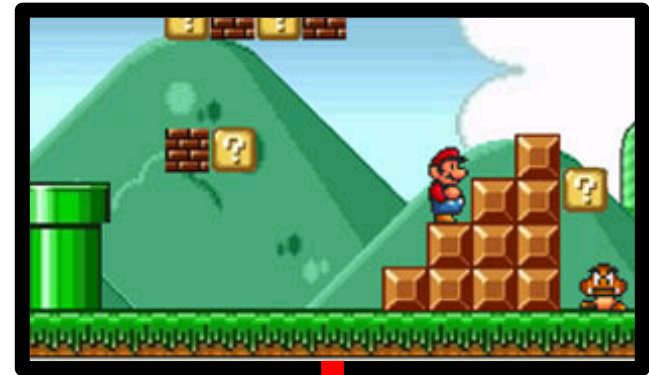
Principe



```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            gauche==true ;  
        }  
}
```



Principe



Gauche = true ;



```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            gauche==true ;  
        }  
}
```

Code correspondant

```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            c.gauche==true ;  
        }  
}
```

```
public class Controle {  
    boolean gauche;  
    boolean droite;  
    boolean saut;  
}
```



Code correspondant

```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            c.gauche==true ;  
        }  
}
```

```
public class Controle {  
    boolean gauche;  
    boolean droite;  
    boolean saut;  
}
```

```
public void update() {  
    if (c.gauche)  
    {  
        mario.x--;  
  
        //gestion des interactions  
        //gestion des collisions  
    }  
}
```



Code correspondant

```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            c.gauche==true ;  
        }  
}
```

```
public class Controle {  
    boolean gauche;  
    boolean droite;  
    boolean saut;  
}
```

```
public void update() {  
    if (c.gauche)  
    {  
        mario.x--;  
  
        //gestion des interactions  
        //gestion des collisions  
    }  
}
```




```
public void keyPressed(KeyEvent e) {  
    if  
        (e.getKeyCode() == e.VK_LEFT)  
        {  
            c.gauche==true ;  
        }  
}
```

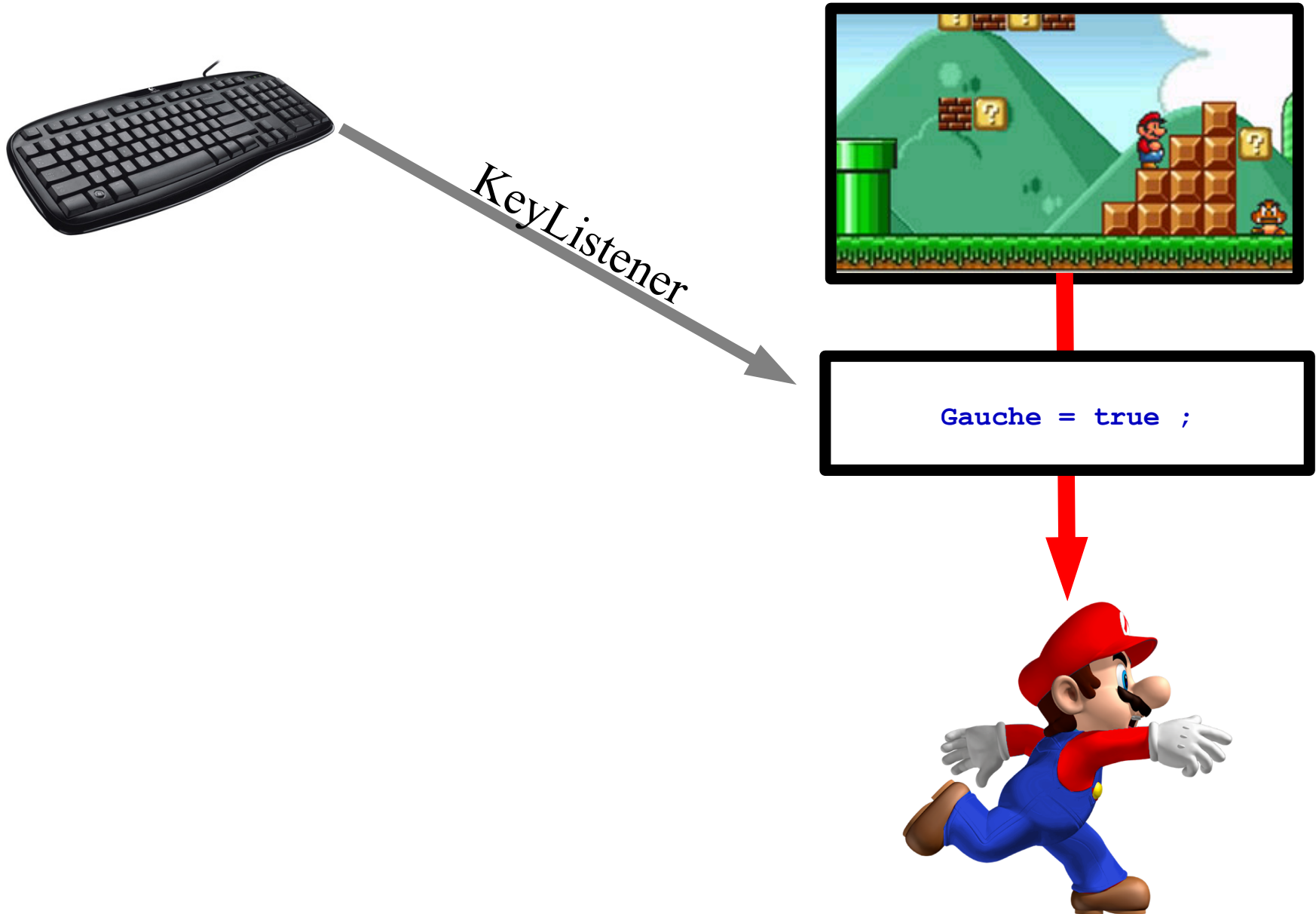
```
public class Controle {  
    boolean gauche;  
    boolean droite;  
    boolean saut;  
}
```

Toujours de la concurrence mais ...

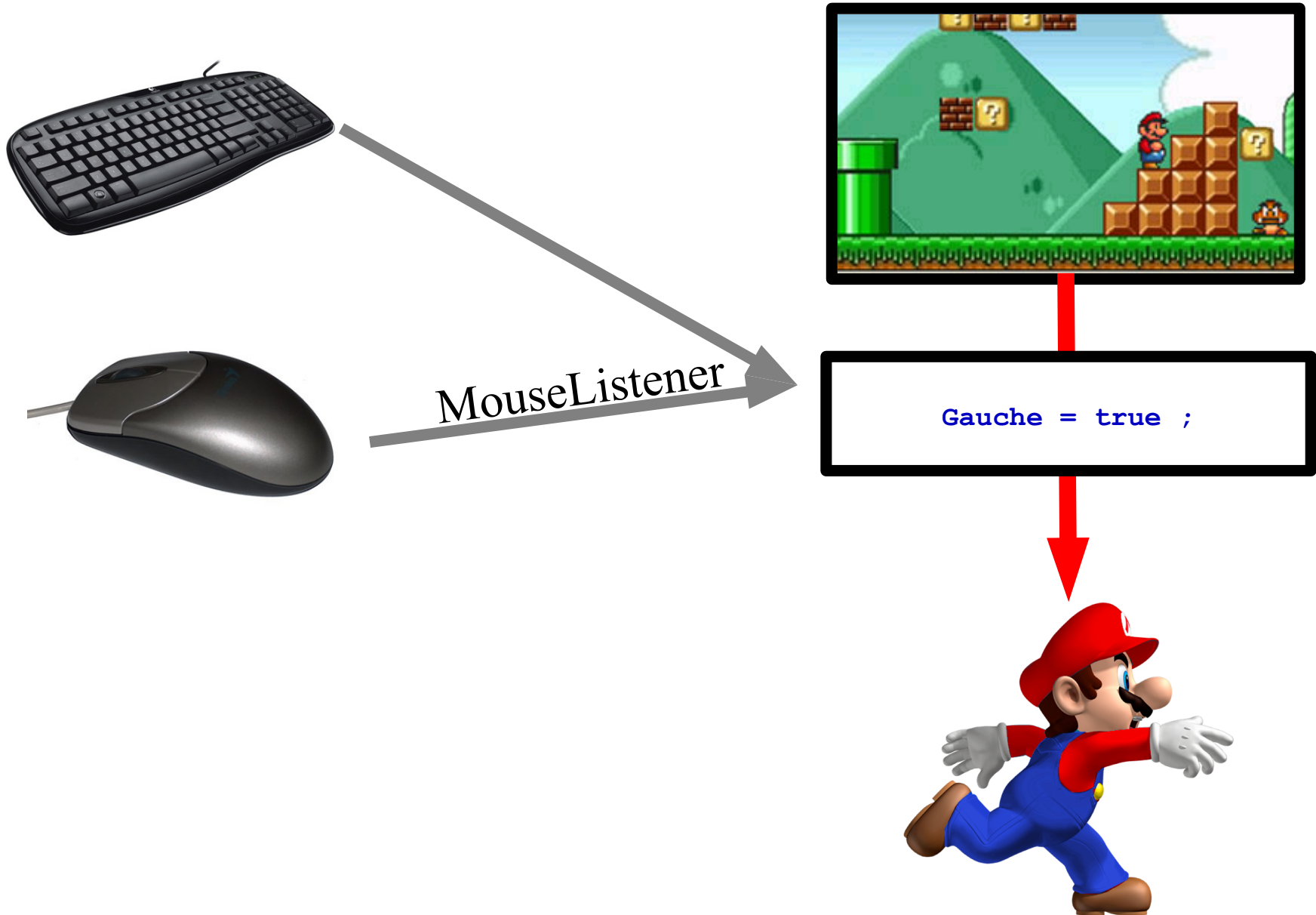
```
//gestion des interactions  
//gestion des collisions  
}
```

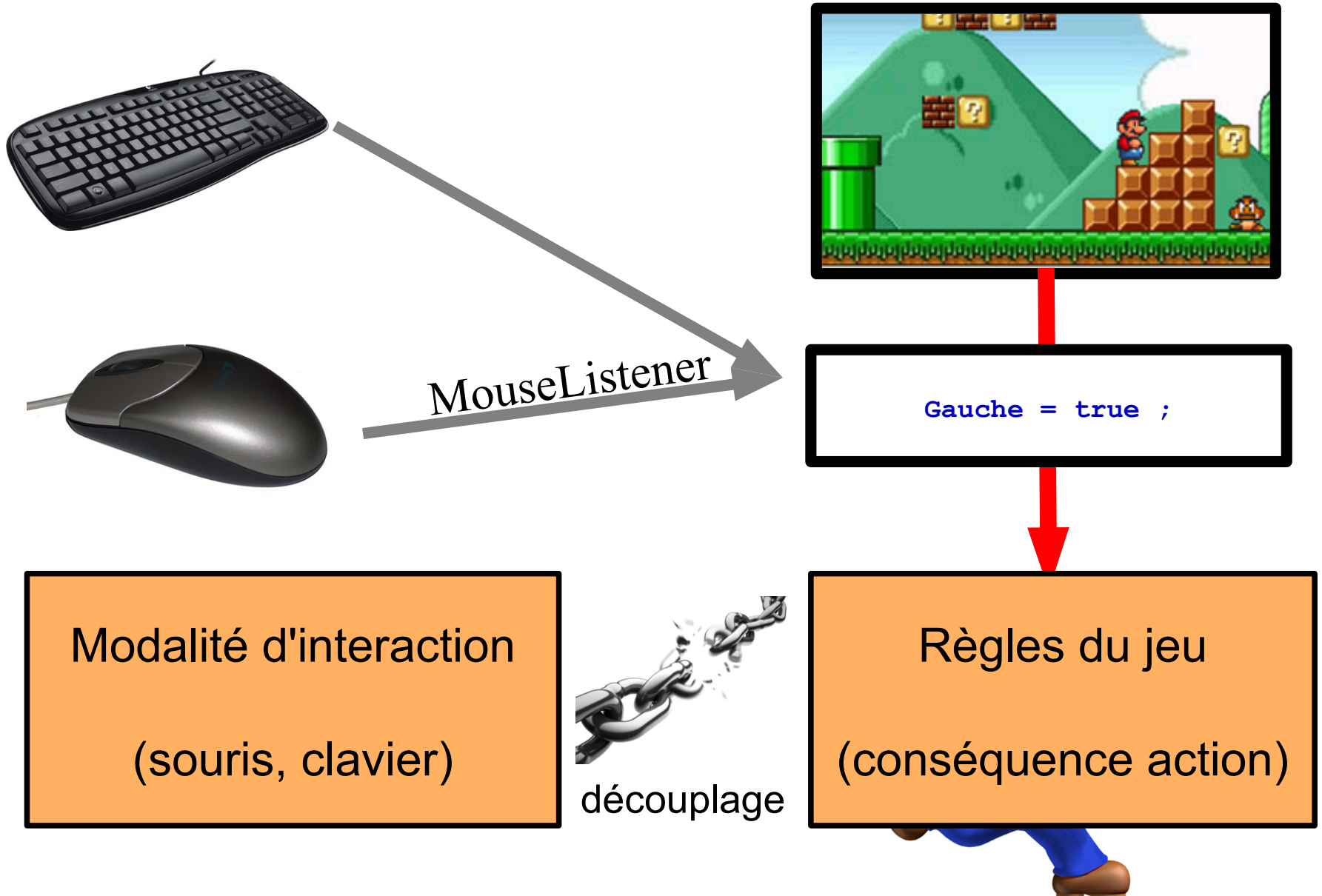
- **Instruction élémentaire = instantané**
 - Modifier un booléen
 - Peu de risques de concurrence d'accès
 - Objets dans état cohérent
- **Interactions centralisées**
 - Lois du monde dans update()
- **Souplesse dans utilisation**
 - Changer de contrôleur

Principe



Principe






Modalité d'interaction
(souris, clavier)



découplage

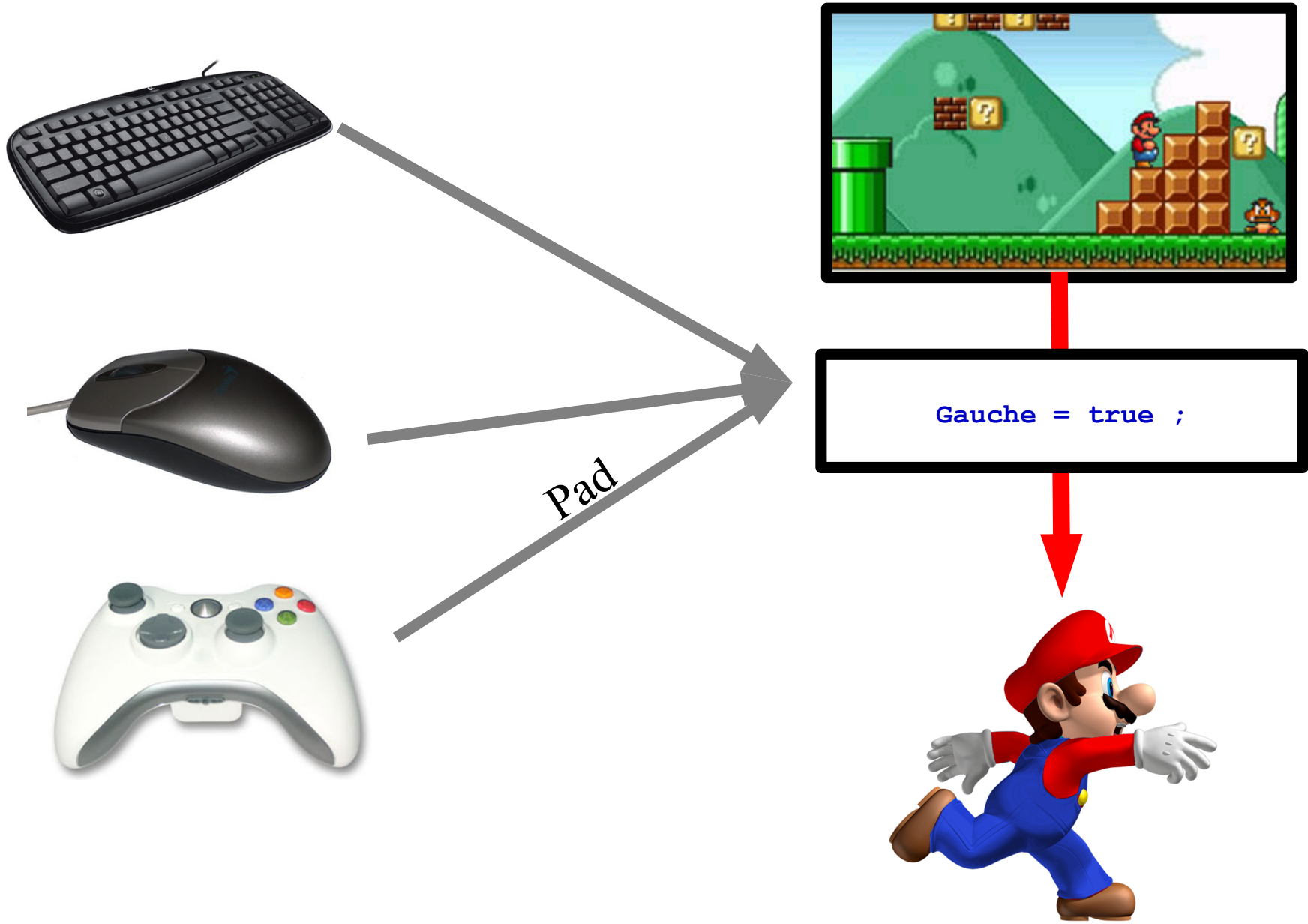
Règles du jeu
(conséquence action)

- **Clavier**
 - Listener
- **Joystick & DDR & PS2 device ...**
 - Librairies (ex : JInput)
- **Webcam (ex eyetoy)**
 - Analyse d'image
- **Wii mote**
 - Librairie existante en java

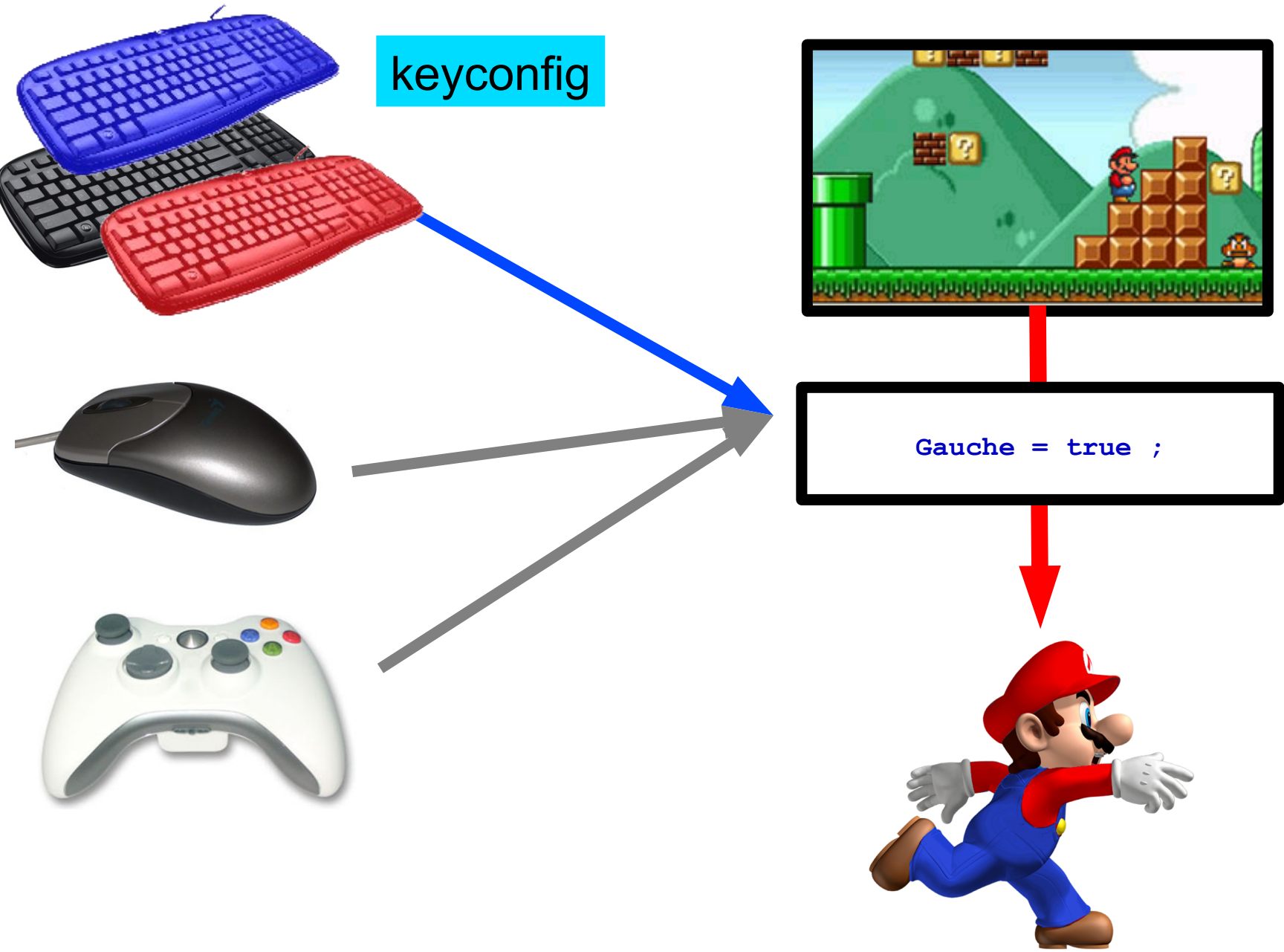


Thread
qui sonde
régulièrement

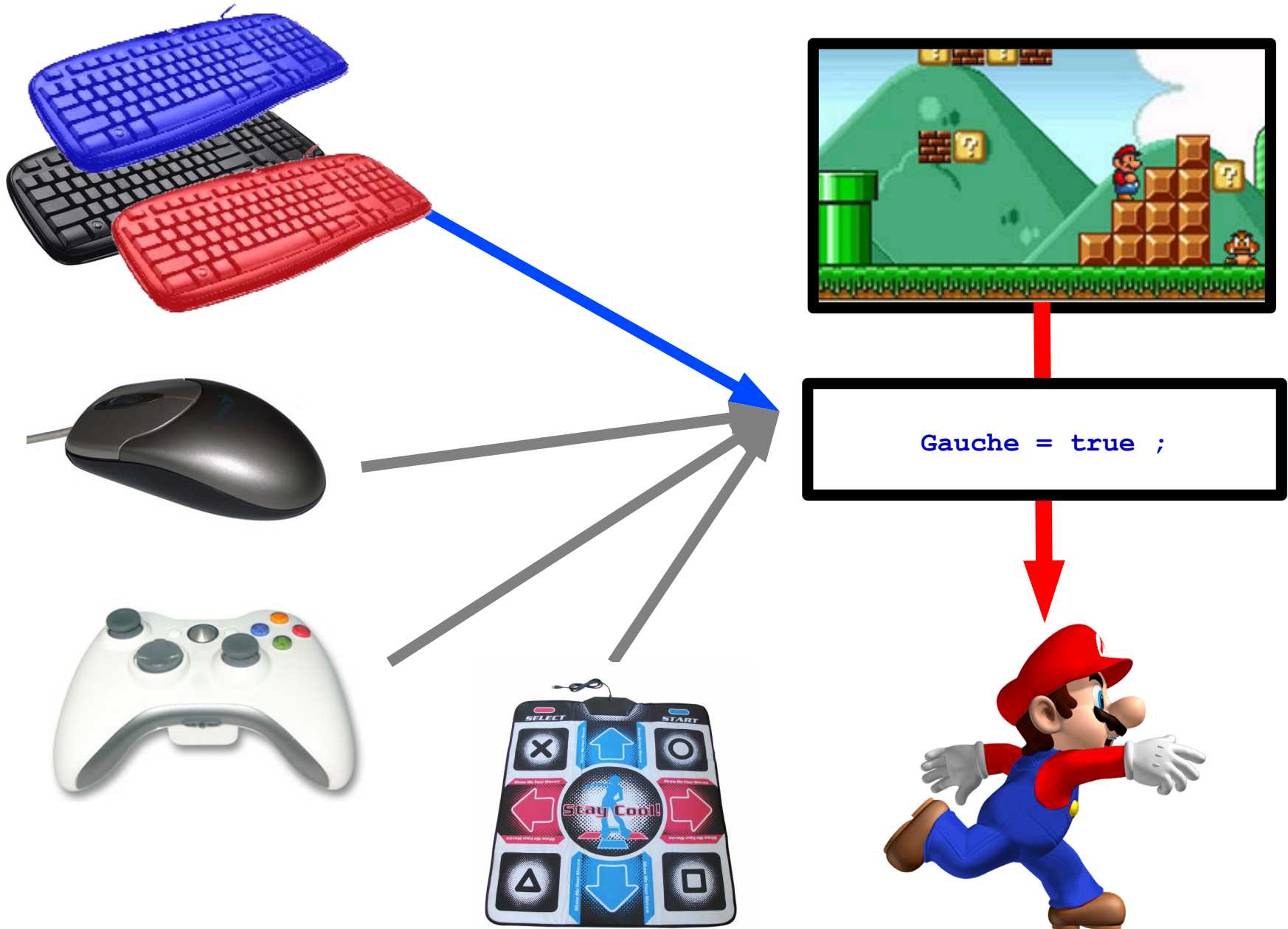
Principe



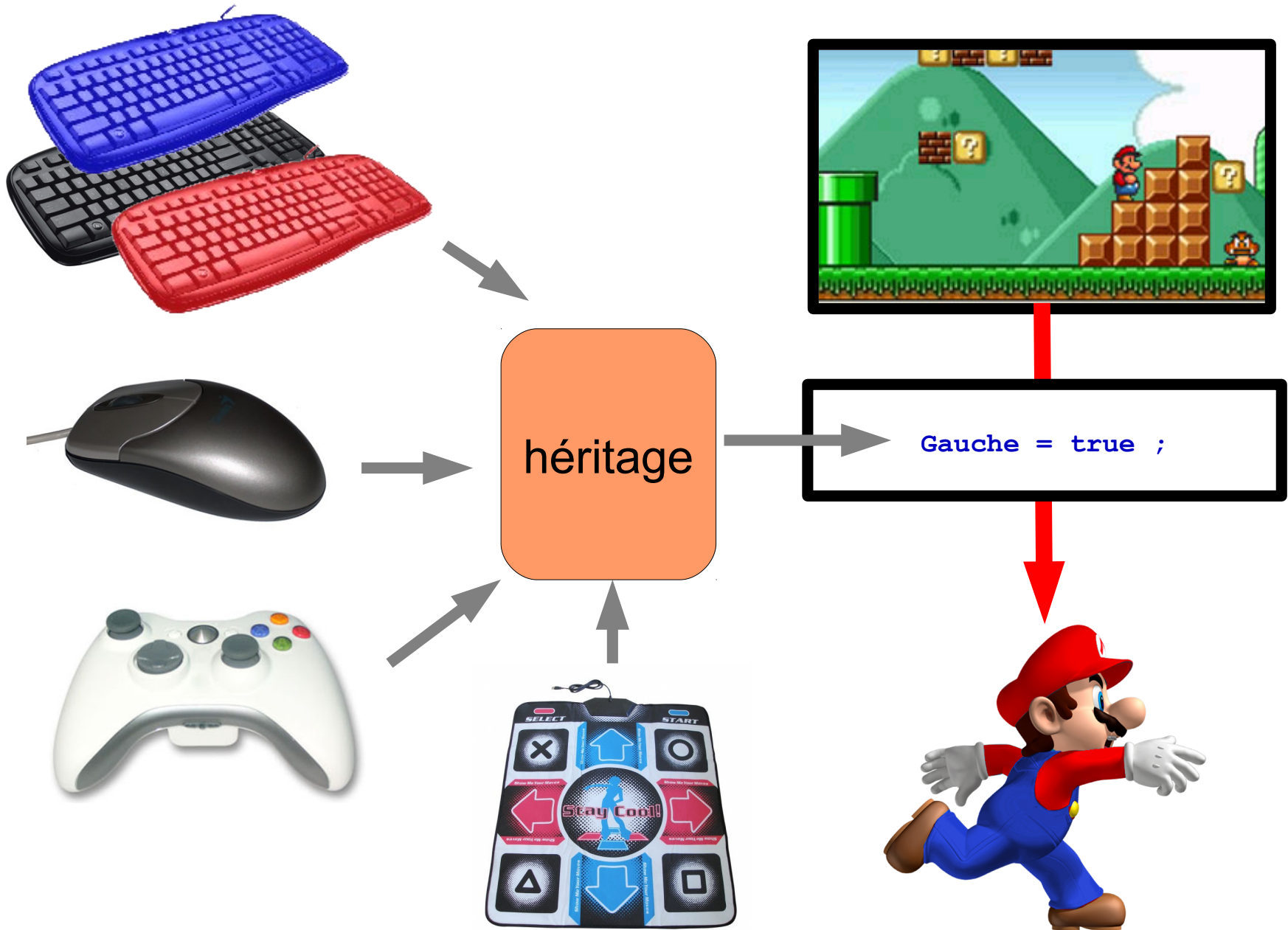
Principe

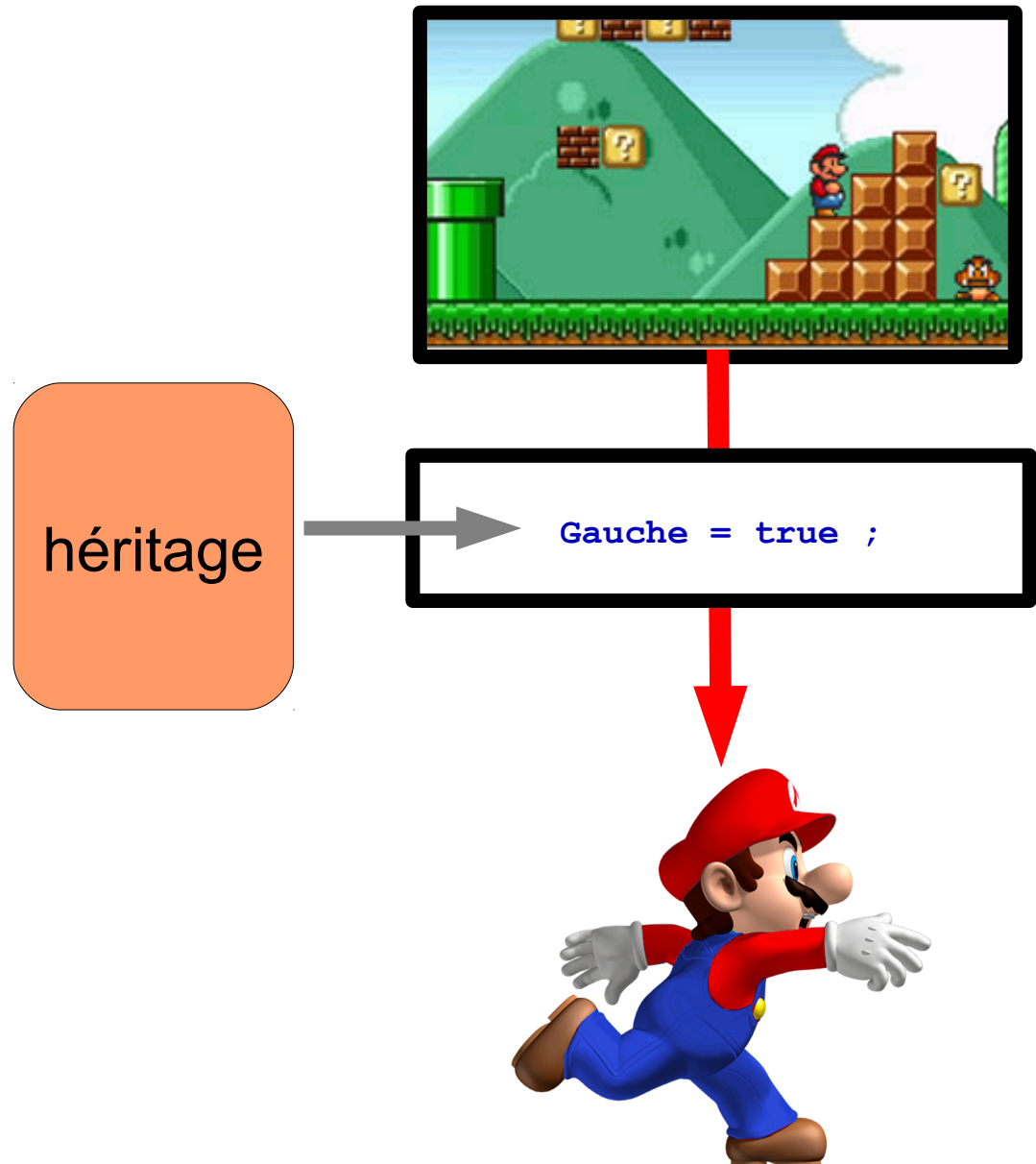


Principe



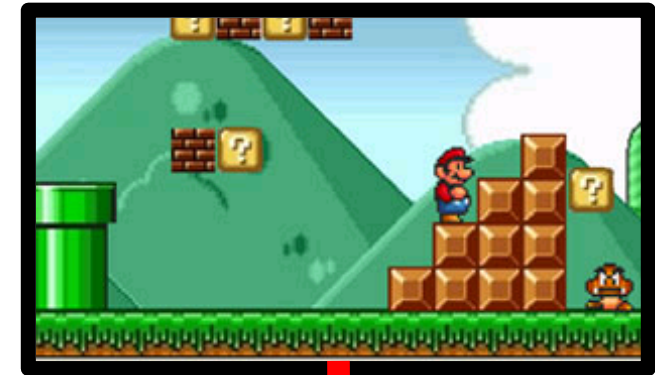
Principe







héritage



```
Gauche = true ;
```



IA Capable de remplacer un joueur
Ex : infinite Mario

Diagramme de classe

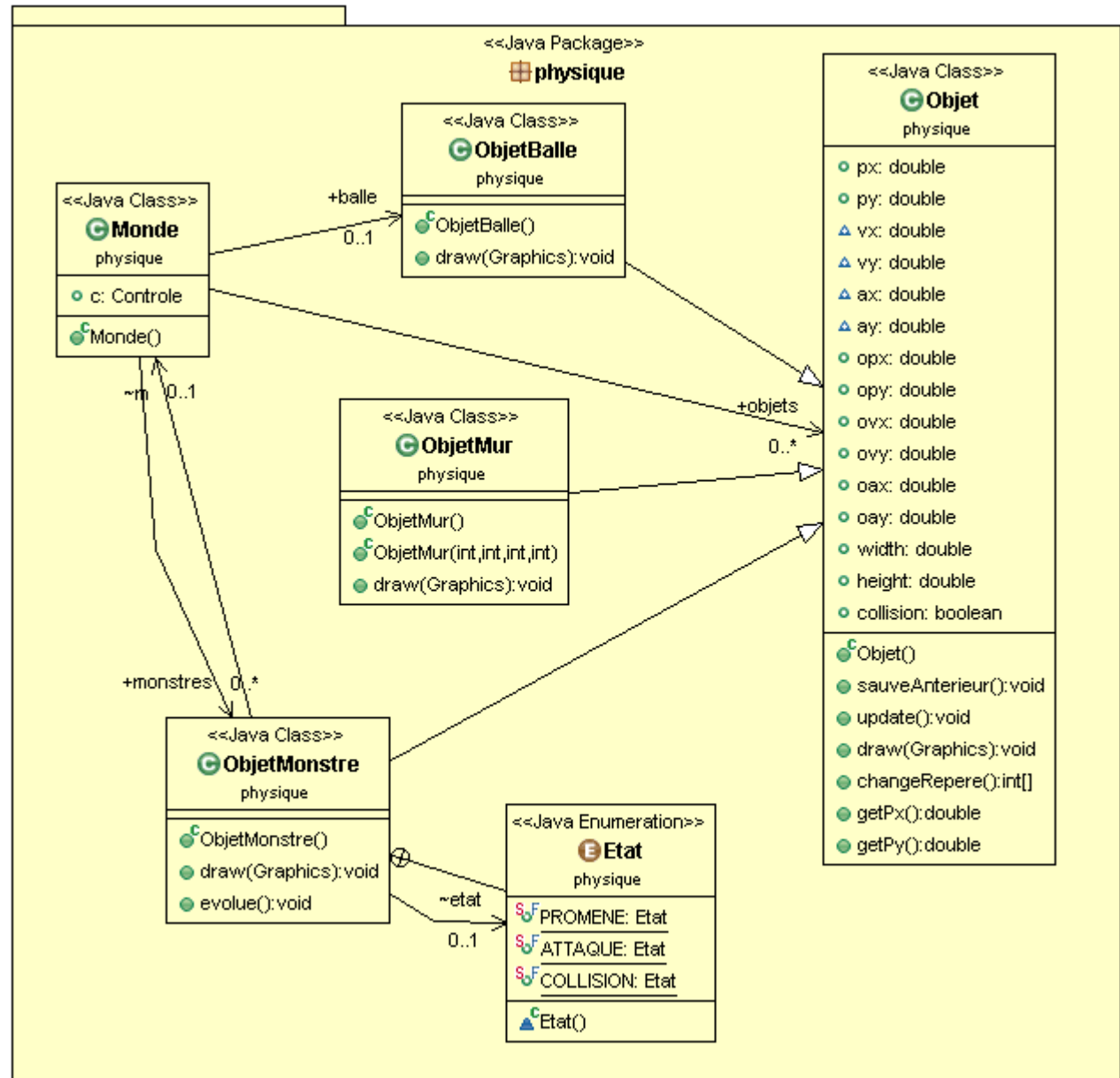


Diagramme de classe

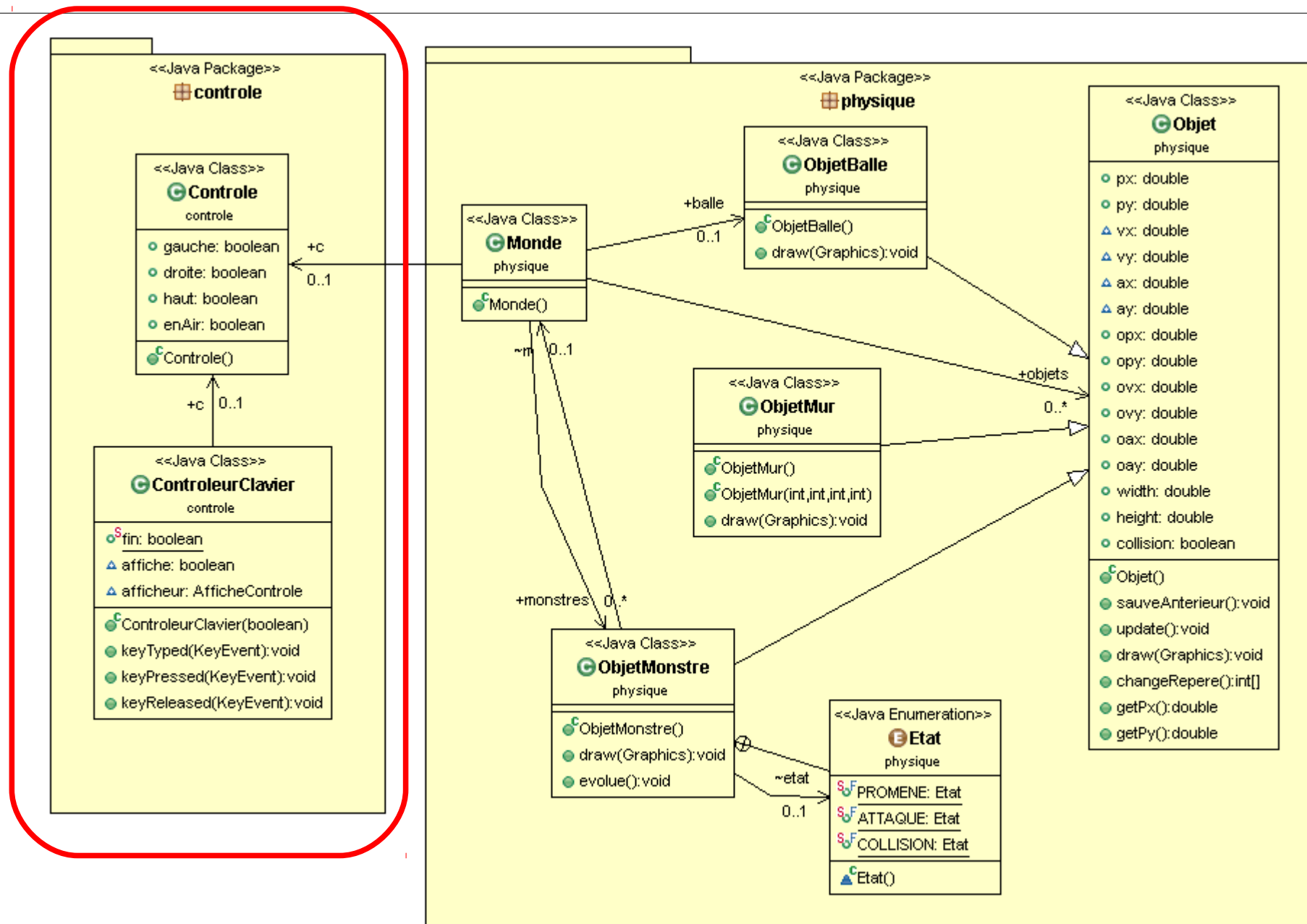


Diagramme de classe

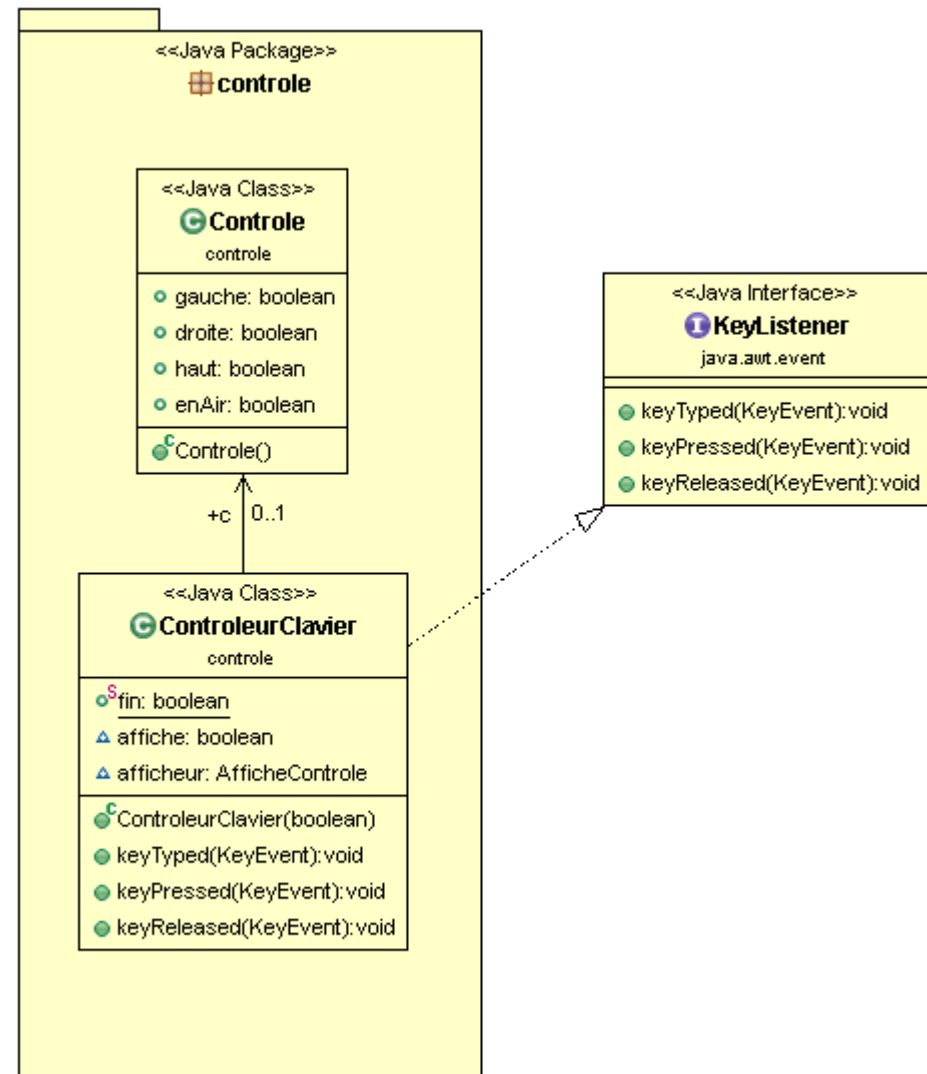
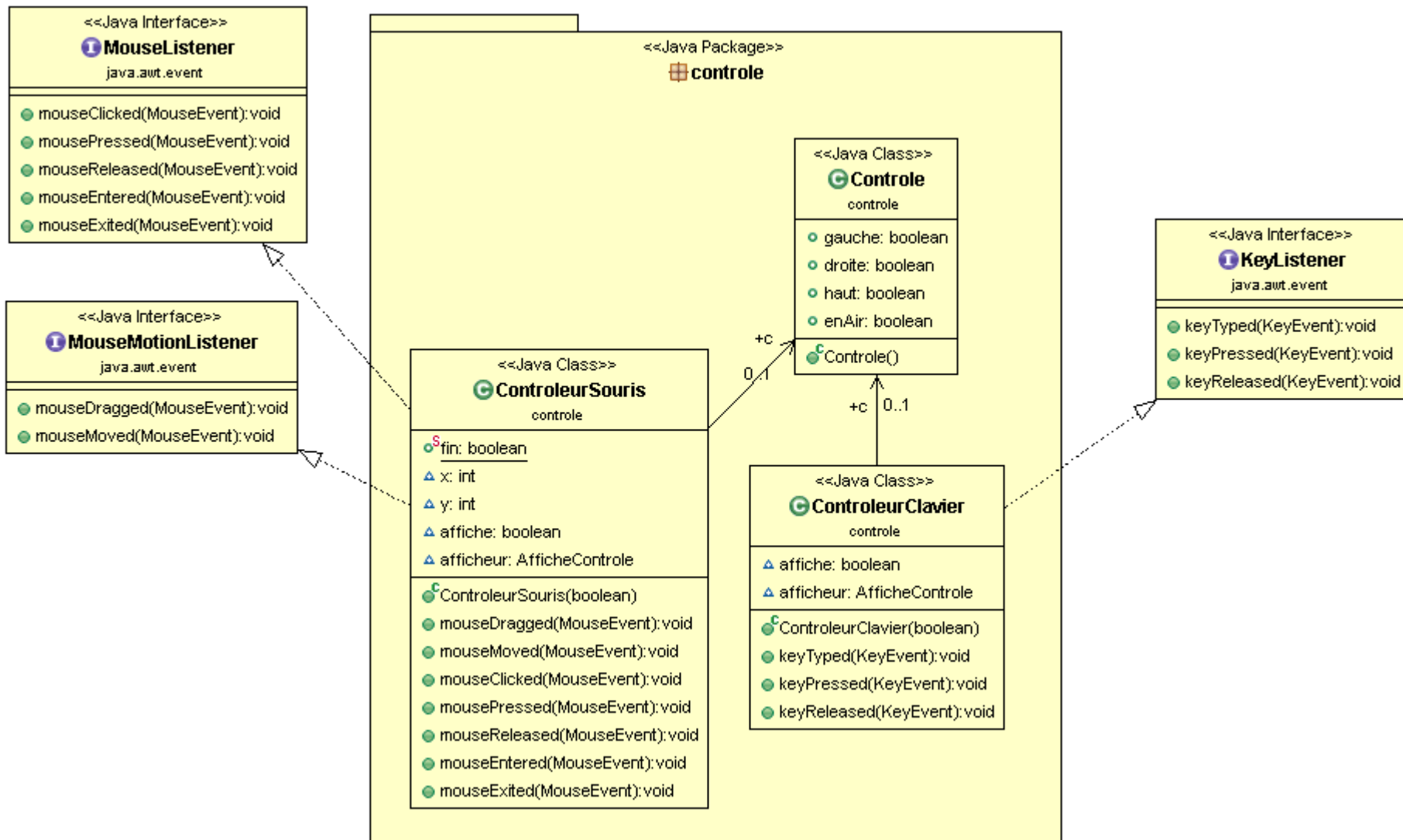


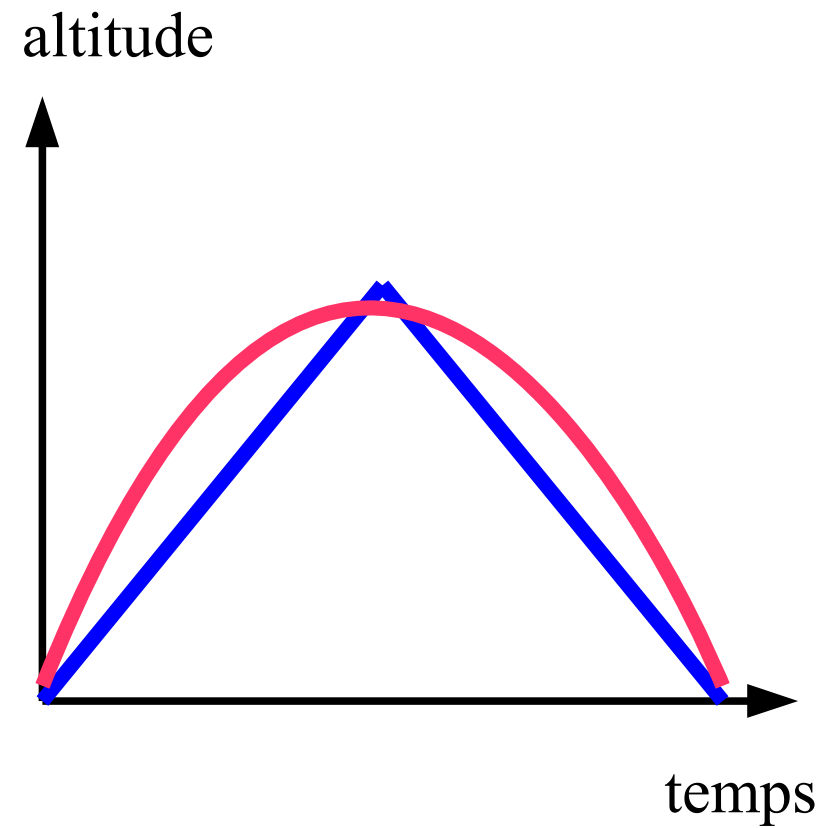
Diagramme de classe



Ajout second contrôleur

- Comment réagir au contrôle ?
- **Mouvement Vertical :**
 - Modifier vitesse verticale
 - La gravité fait retomber

- Comportement de saut



- Comment réagir au controle ?

- **Mouvement Vertical :**

- Modifier vitesse verticale
- La gravité fait retomber

```
Perso.vy = cste
```

- **Mouvement Horizontal :**

- Vitesse constante
- Modifier x

```
Perso.x = x + cste
```

- Comment réagir au controle ?

- **Mouvement Vertical :**

- Modifier vitesse verticale
- La gravité fait retomber

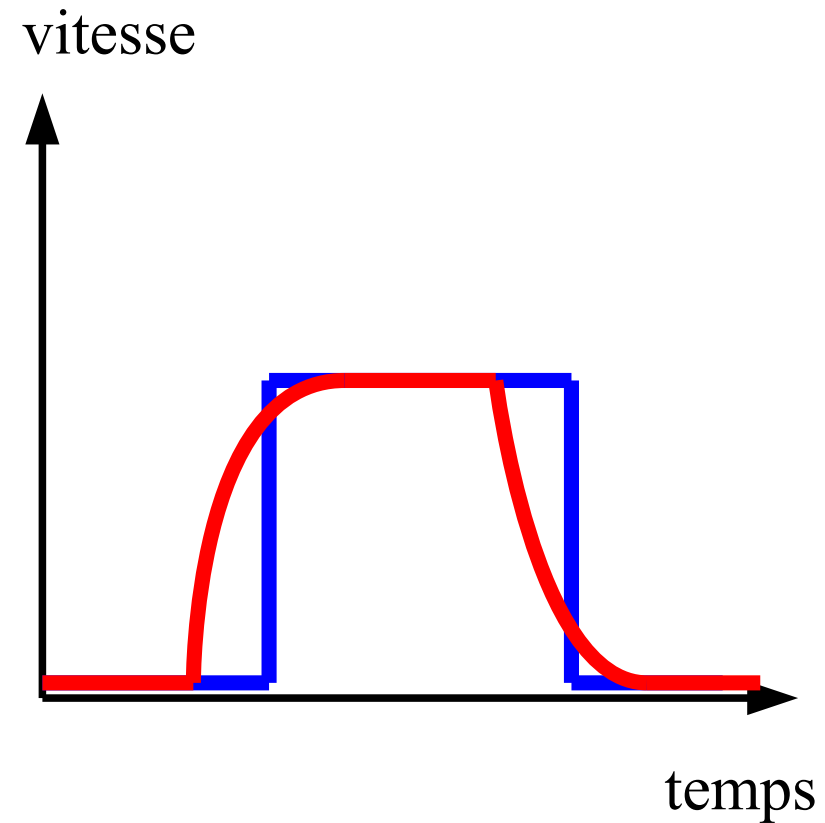
$$\text{Perso.vy} = \text{cste}$$

- **Mouvement Horizontal + complexe :**

- Vitesse constante
- Modifier x

$$\text{Perso.ax} = \text{cste}$$

- Course



- Modèles du monde

Démonstration Partie 6

Mise en œuvre d'un contrôleur

Clavier / Souris / IA

06x01 – affichage du contrôleur

06x02 – contrôle = vitesse horizontale

06x03 – contrôle = saut + affiche collisions

06x04 – contrôle = saut + sans afficher

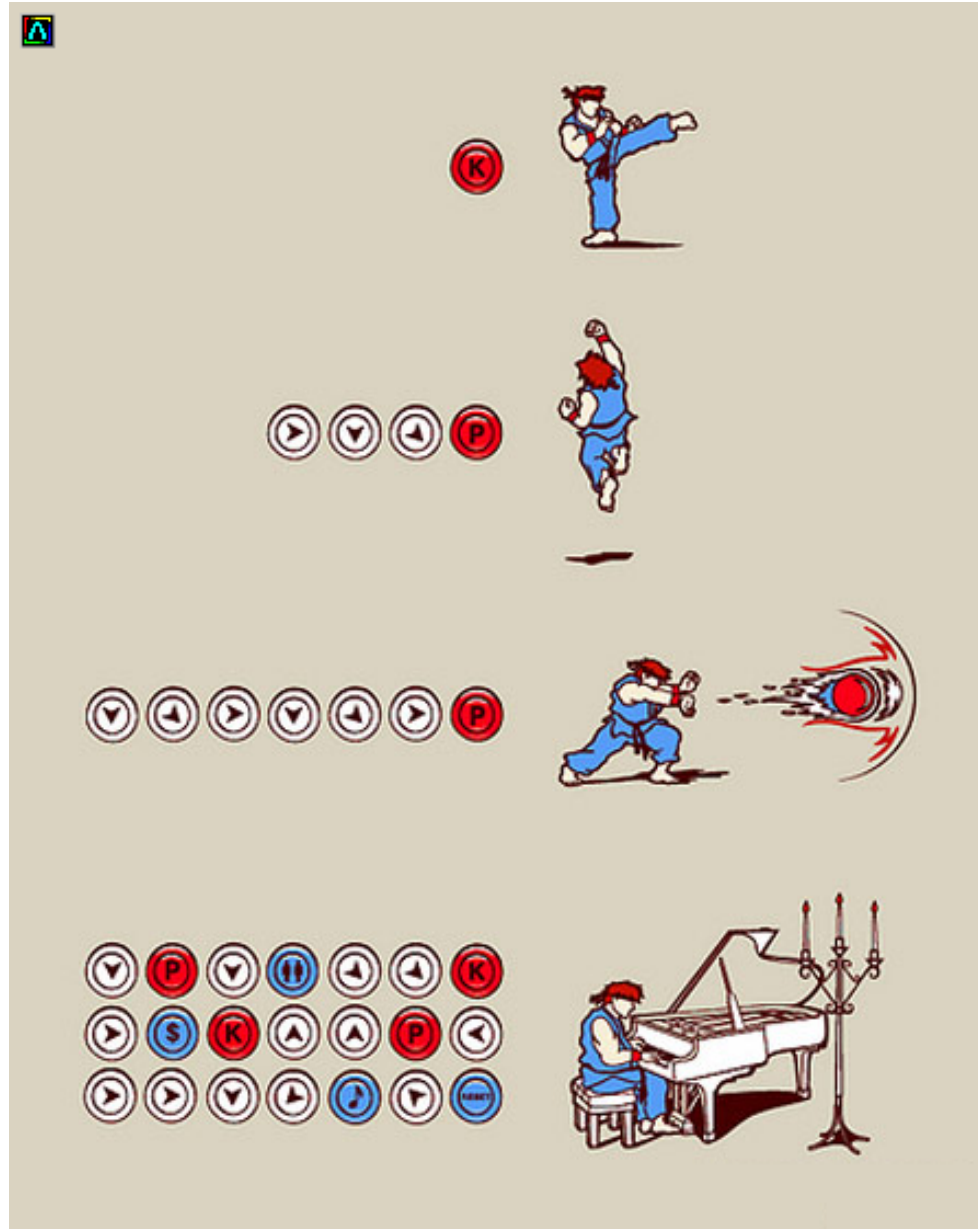
06x05 – contrôle souris

06x06 – contrôle horizontal = accélération

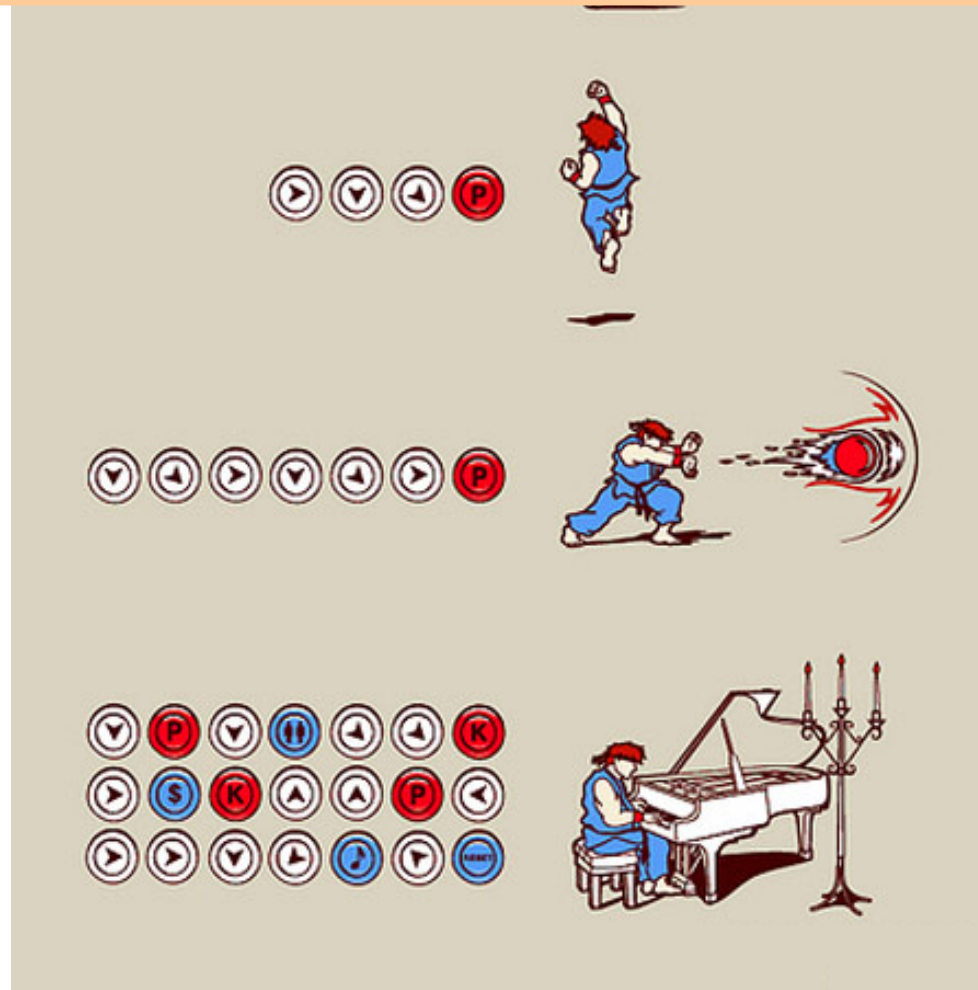
- **En java**
 - Librarie Jinput
 - Reconnais les périphériques
- **Fonctionnement**
 - Récupérer une liste de contrôleurs
 - Analyser chacun des actionneurs du contrôleur

06x07 – gestion joypad

Contrôleurs plus subtils



Utiliser des automates pour reconnaître



- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
- Réseau

Diagramme de classe

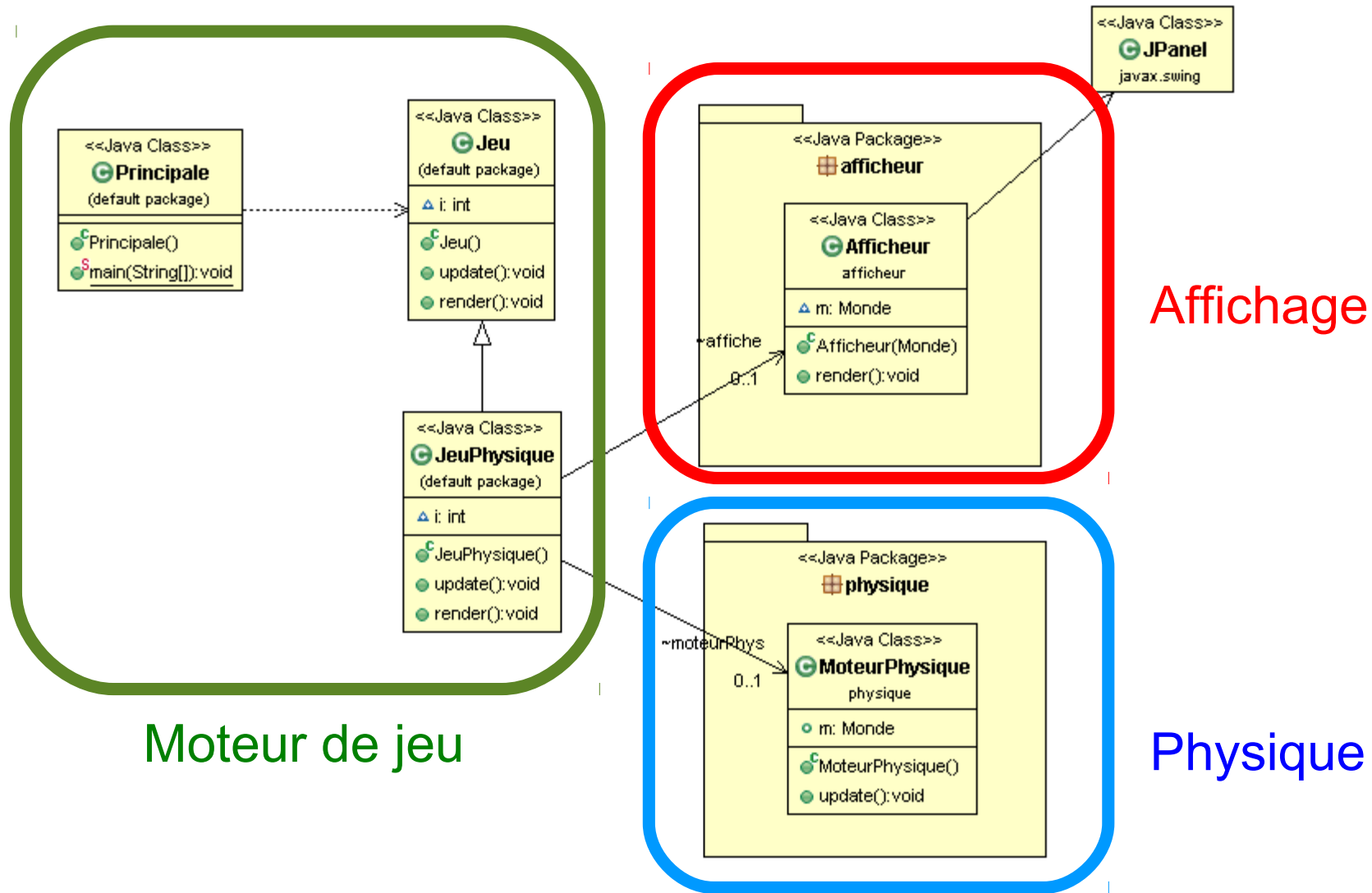
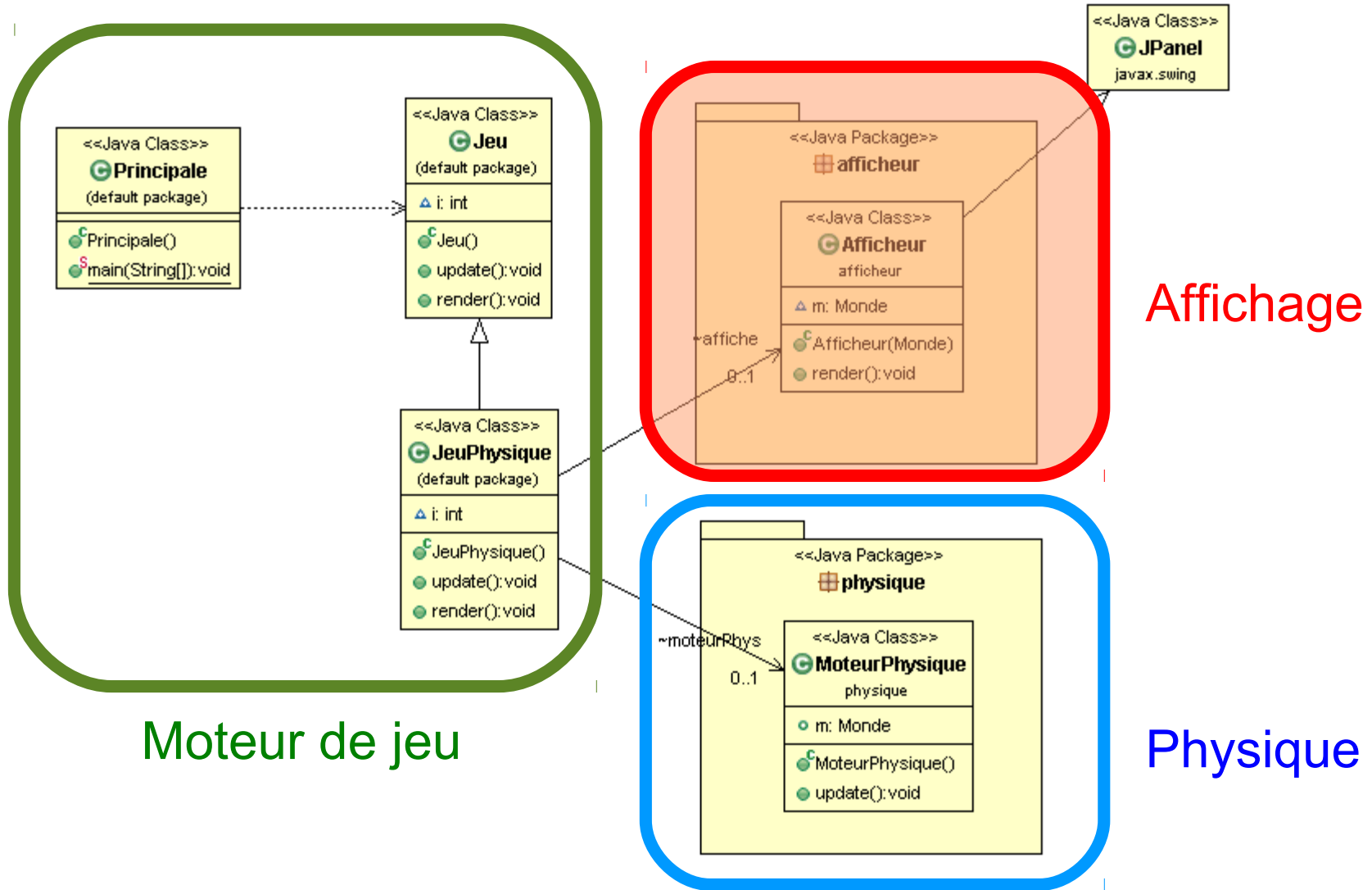


Diagramme de classe



- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
 - Vue subjective / changement de repère
 - Double buffering
 - Sprites et animation
 - Scrolling
- Réseau

- Coeur de l'affichage
 - L'affichage **NE MODIFIE PAS** les données
 - La vue s'adapte aux données (pas l'inverse)

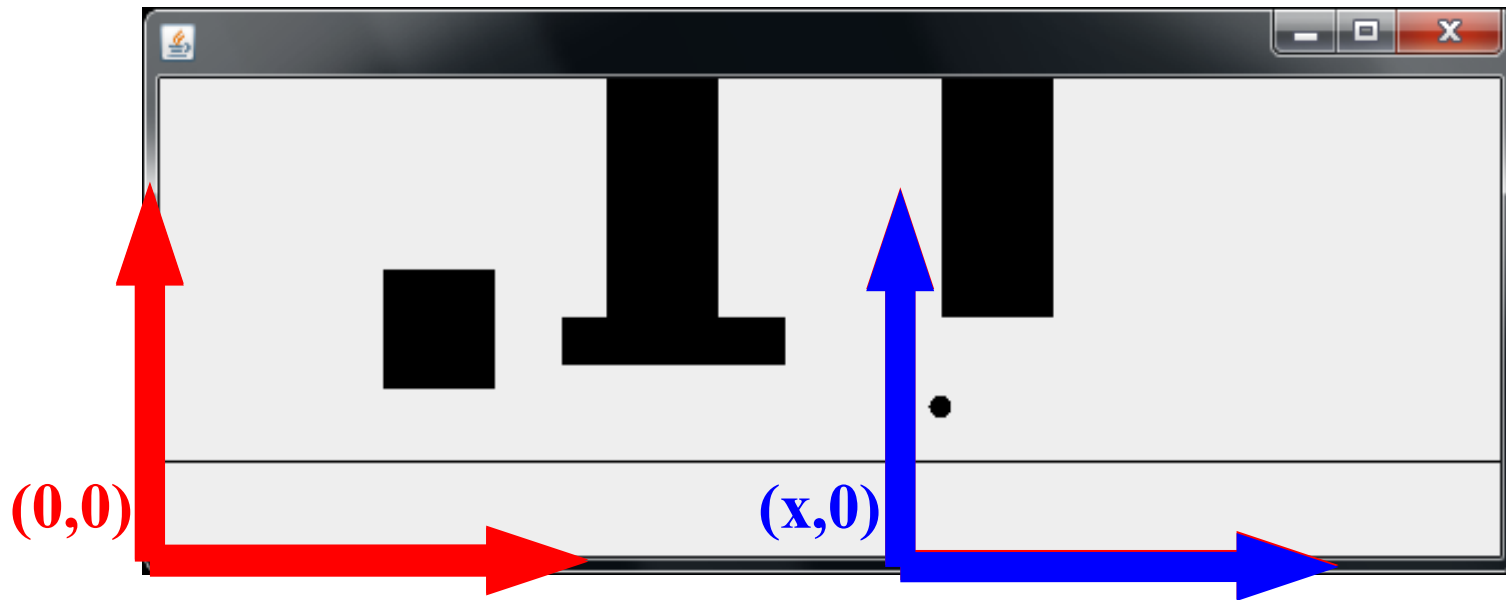
- Exemples :
 - la vue subjective
 - la notion de zoom

- Juste en modifiant QUELQUES caractères (10)



07x01 – vue subjective

- Juste en modifiant QUELQUES caractères (10)



Principe \Rightarrow changer de repere

- Juste en modifiant QUELQUES caractères (10)



Juste dans l'affichage

**Ne pas changer les données !!!
Elles contiennent l'information utile**

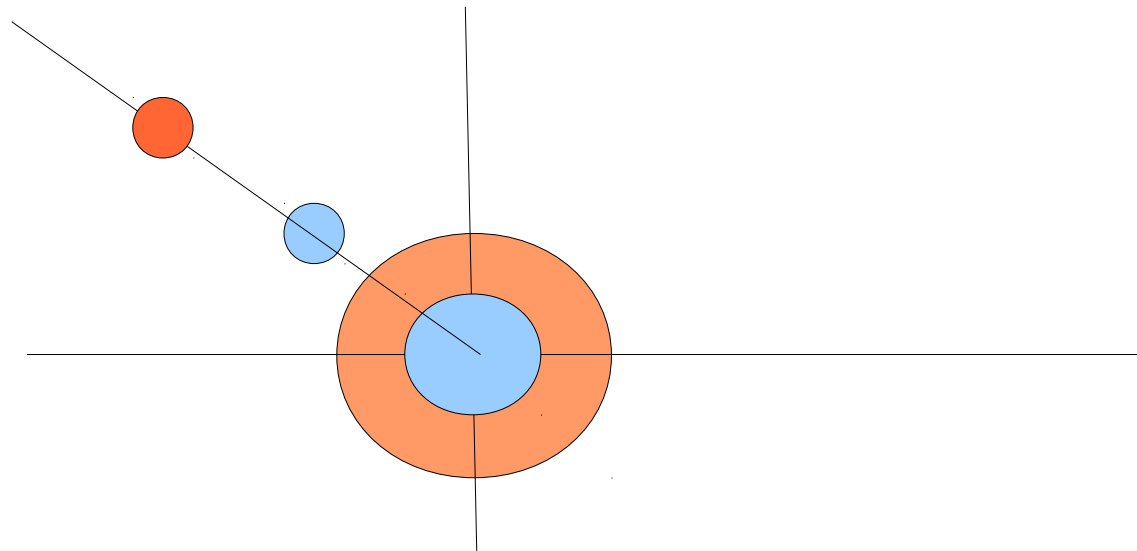
Ex projet tut où les objets se décalaient

cipe ⇒ changer de repere

Vue subjective

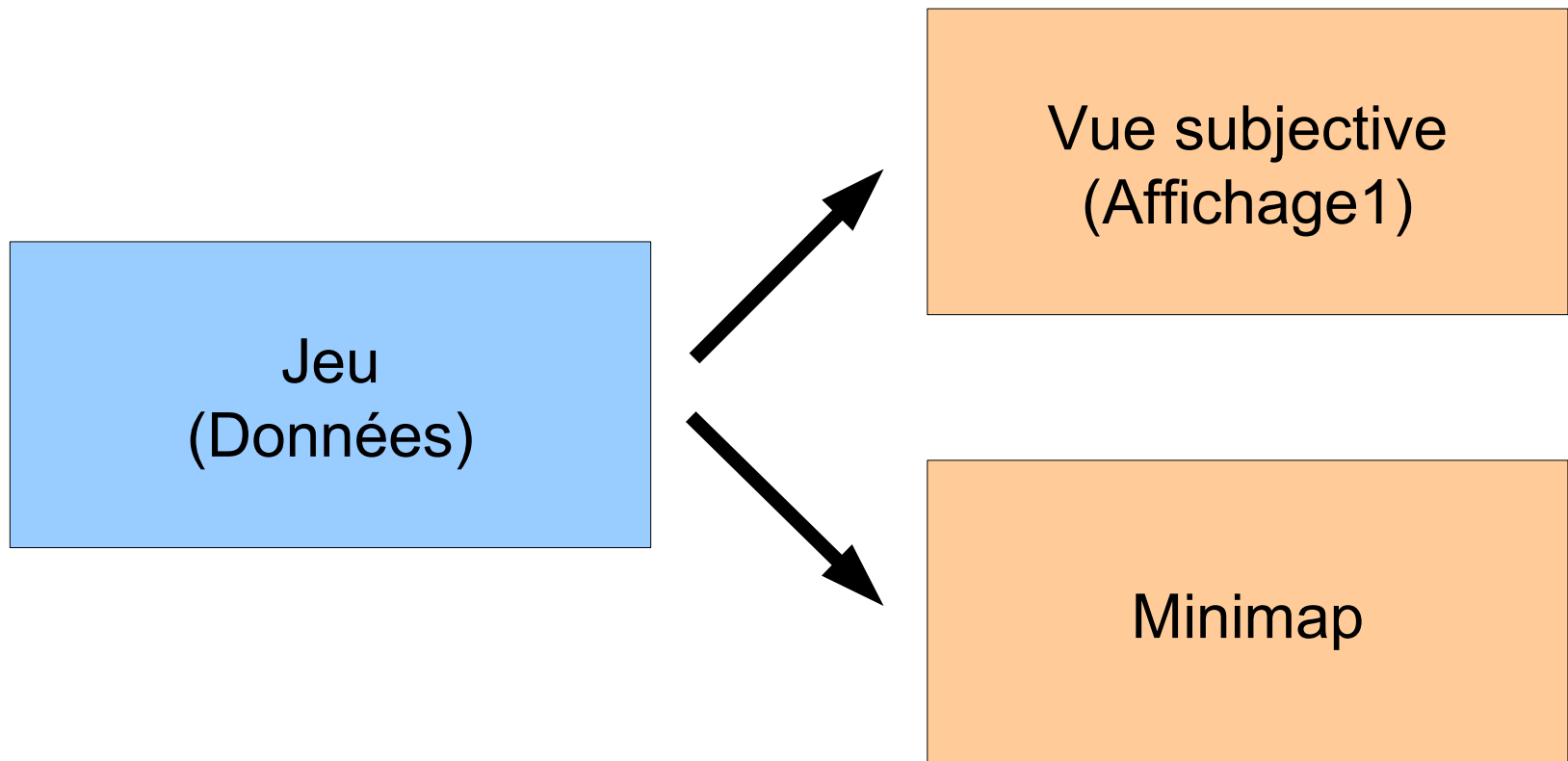
- Juste en modifiant QUELQUES caractères (10)
- Vue globale dans render
 - La balle affichée en (x,y)
 - Chaque objet o est affiché en (x_o,y_o)
- Vue subjective **dans render**
 - **Décaler le repère $(200-x, 0)$**
 - Balle affichée en $(x+200-x, y+0) = (200,0)$ fixe
 - Objet affiché en $(x_o+200-x, y_o)$

- **Même principe**
 - Faire une transformation espace réel => écran
 - Homothétie centrée sur personnage

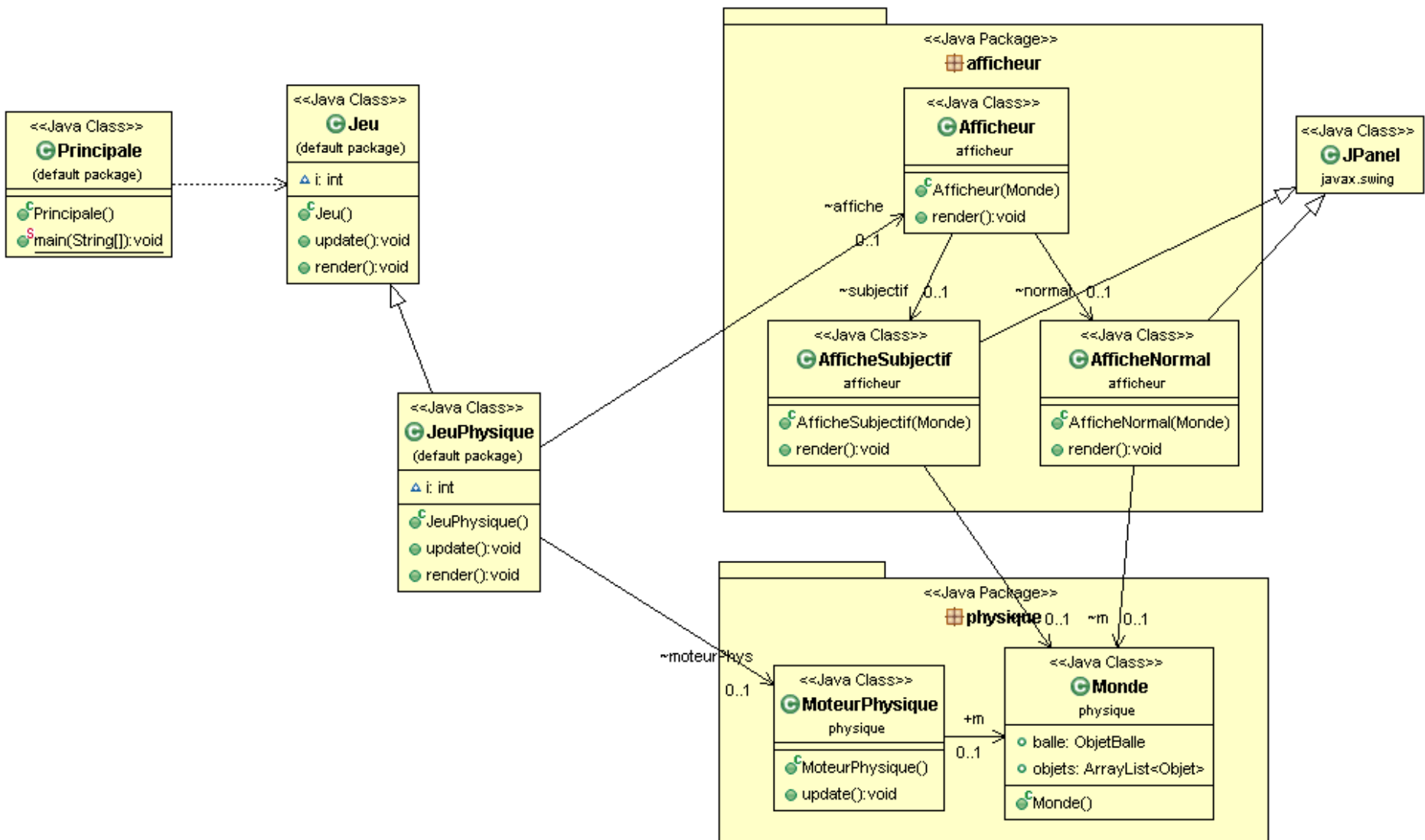


Démonstration plus tard

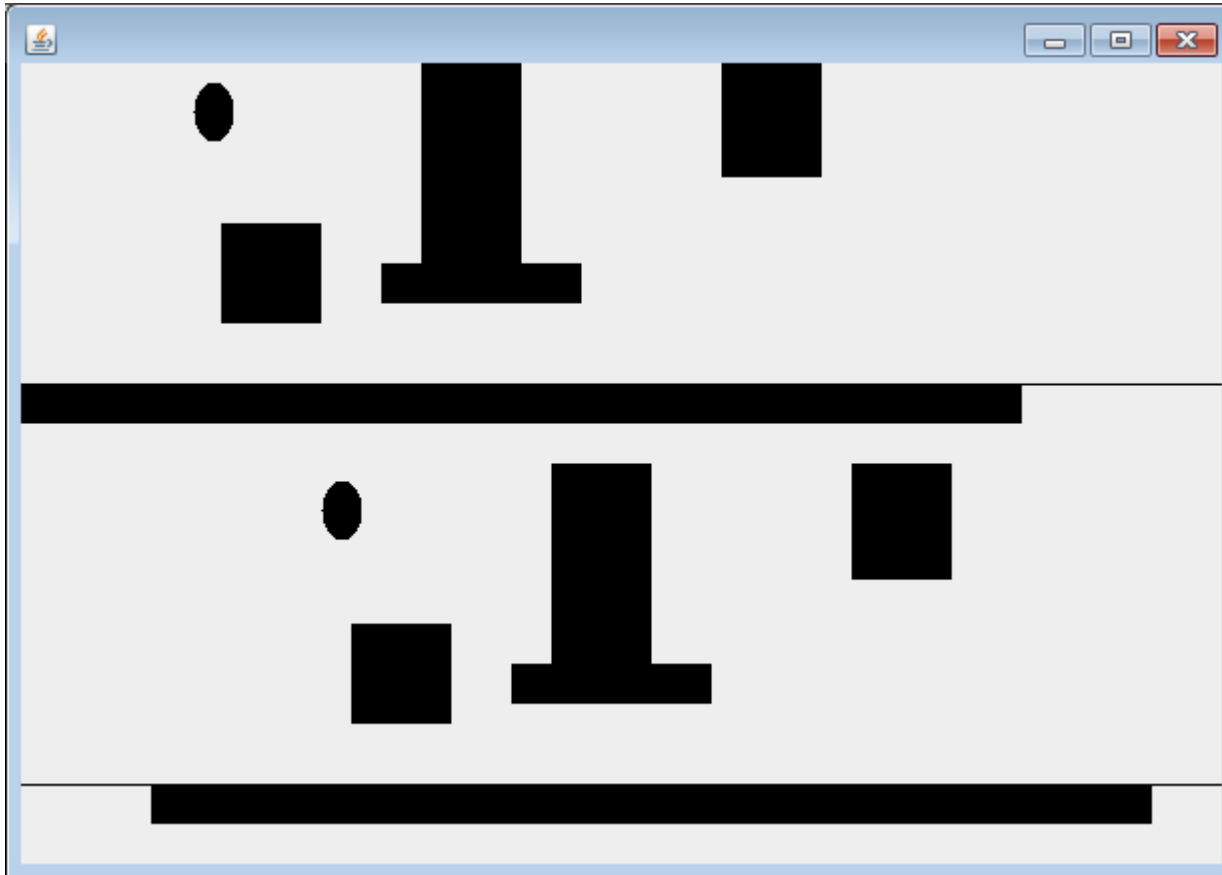
- Memes données pour des vues différentes
 - Principe architecture MVC



Demonstration multi-affichage



Démonstration multi-affichage



- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
 - Vue subjective / changement de repère
 - Double buffering
 - Sprites et animation
 - Scrolling
- Réseau

- **Avec repaint()**
 - Ne sait pas quand fini
 - Transformer repaint en une méthode
- **Active rendering**
 - Affichage plus dans repaint()
 - Méthode render
 - Récupère le graphics de l'image
 - Dessine dans le Jpanel

Active rendering

- Active rendering
 - Méthode render

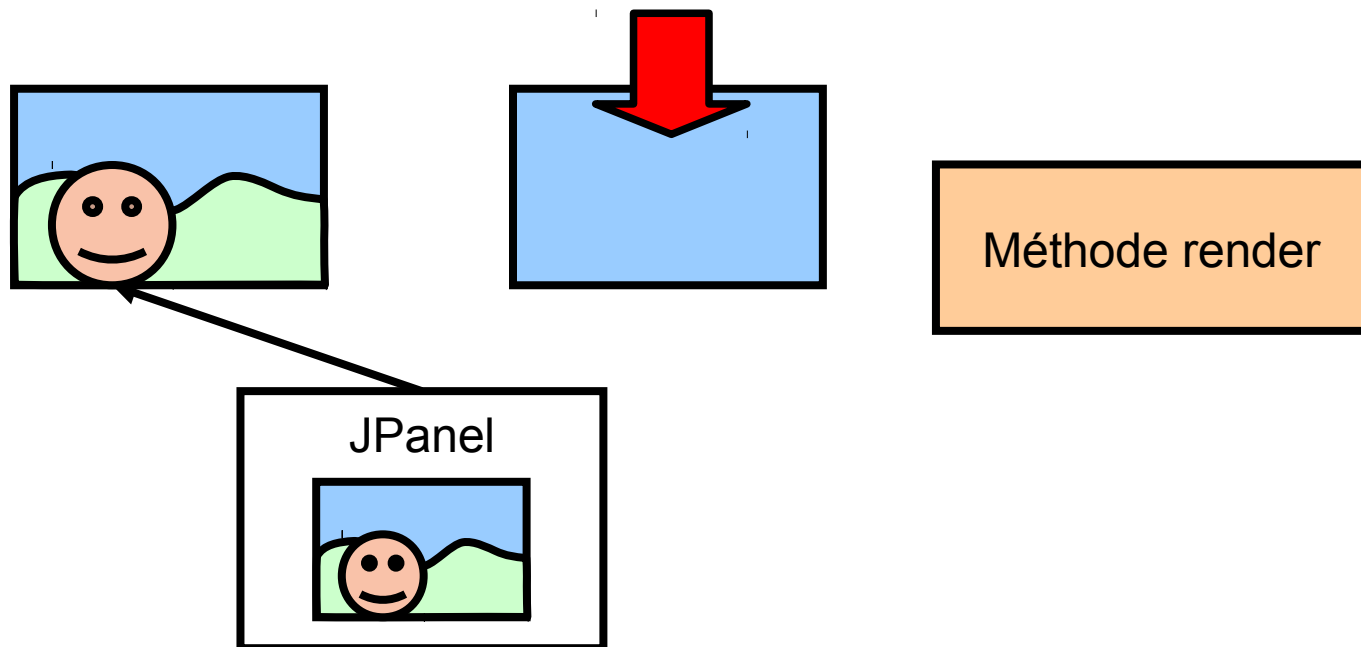
```
public void render()  
{  
    Graphics g=p.getGraphics();  
    g.clearRect(0, 0, 600, 200);  
    g.drawLine(0, 150, 600, 150);  
    g.drawOval((int)px, 150-10-(int)py, 10, 10);  
}
```

- Plus de repaint()

- **Actuellement**
 - Dessiner sur Jpanel affiché
- **Problemes de surbrillance**
 - Affiche surtout image incomplète
 - Synchronisation verticale
- **Principe double buffering**
 - Préparer image de manière cachée
 - Afficher d'un coup

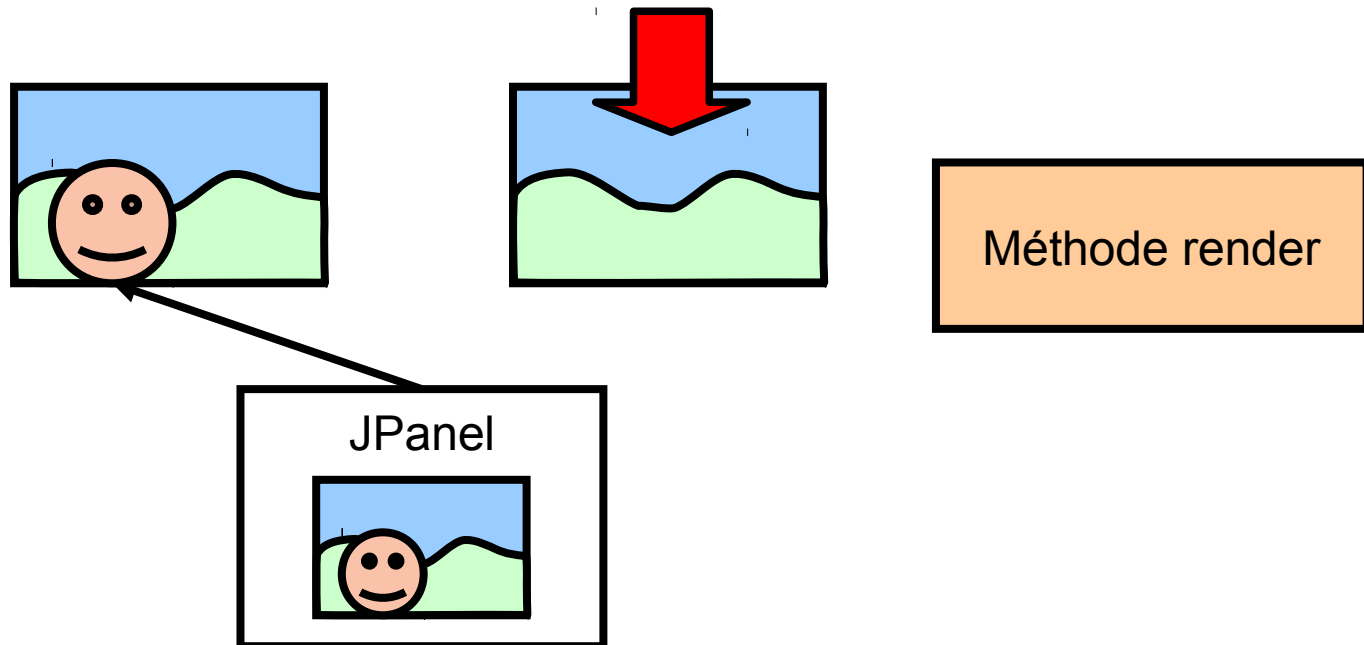
Double buffering

- Principe « Double buffering »
 - Deux images
 - Image affichée et image de travail
 - Repaint inverse les images (copie mémoire rapide)



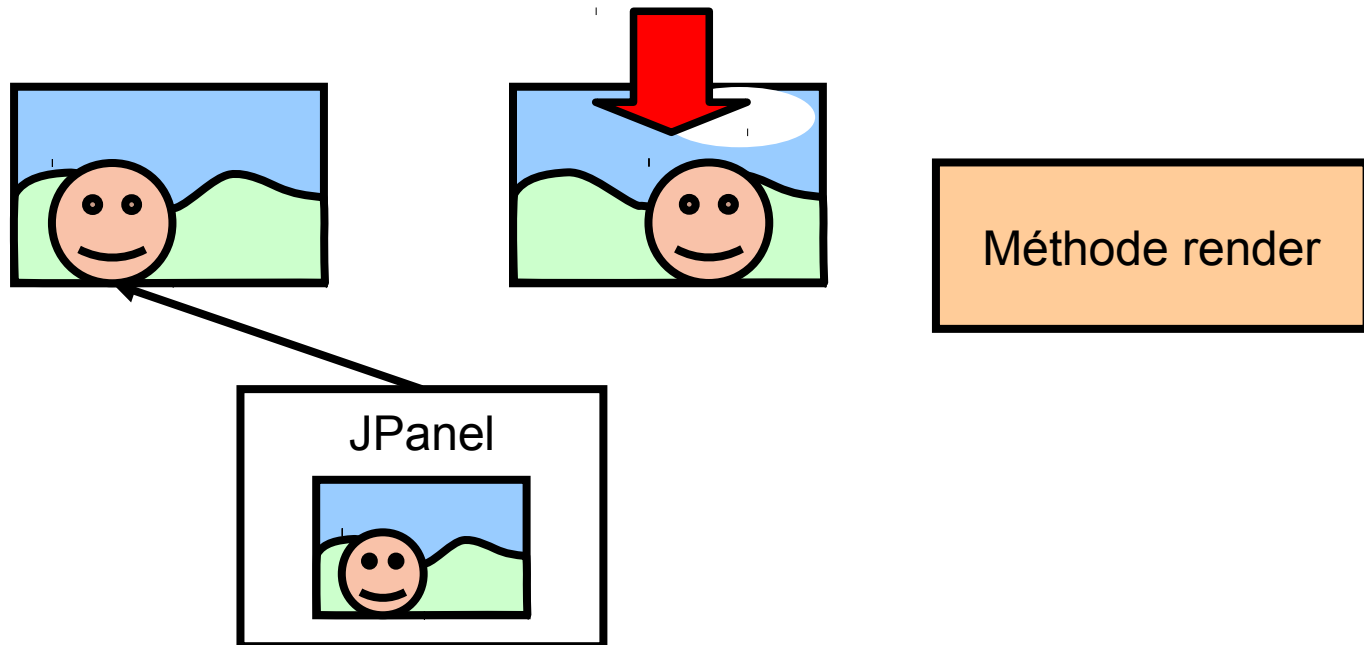
Double Buffering

- Principe « Double buffering »
 - Deux images
 - Image affichée et image de travail
 - Repaint inverse les images (copie mémoire rapide)



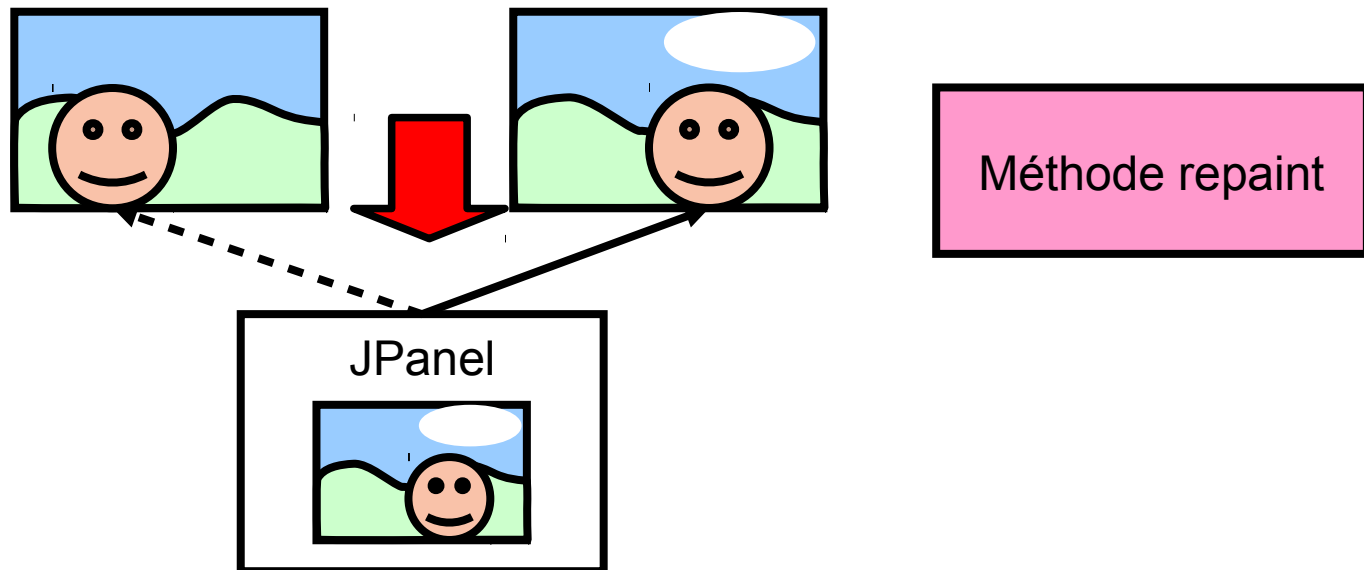
Double buffering

- Principe « Double buffering »
 - Deux images
 - Image affichée et image de travail
 - Repaint inverse les images (copie mémoire rapide)



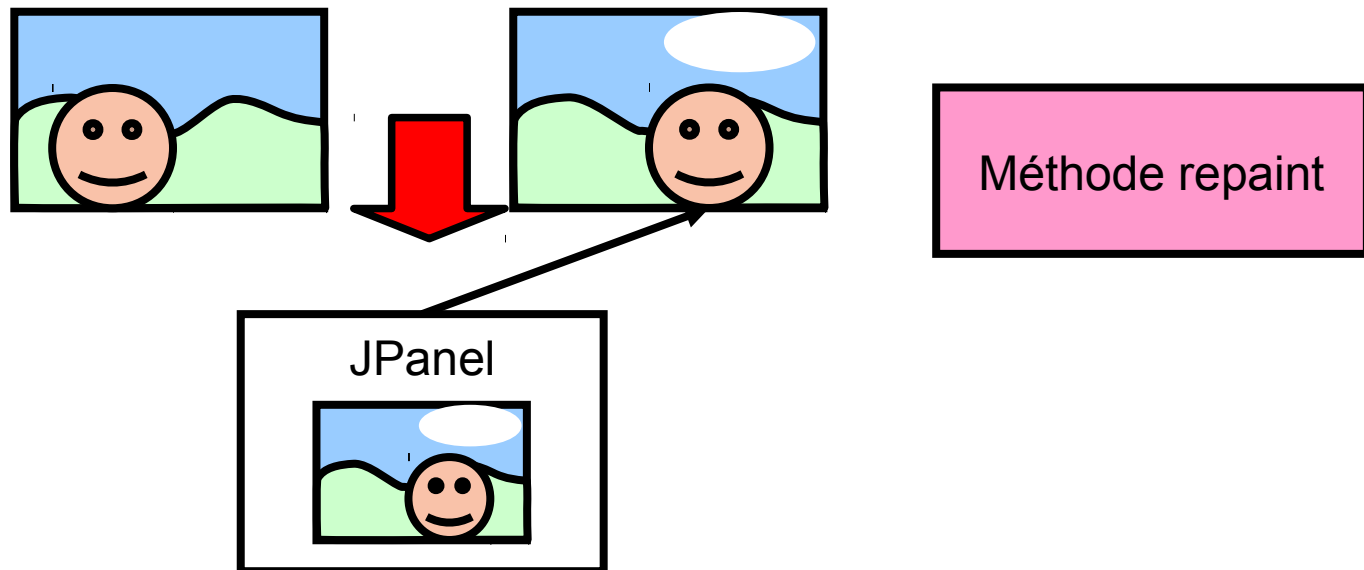
Double buffering

- Principe « Double buffering »
 - Deux images
 - Image affichée et image de travail
 - Repaint inverse les images (copie mémoire rapide)



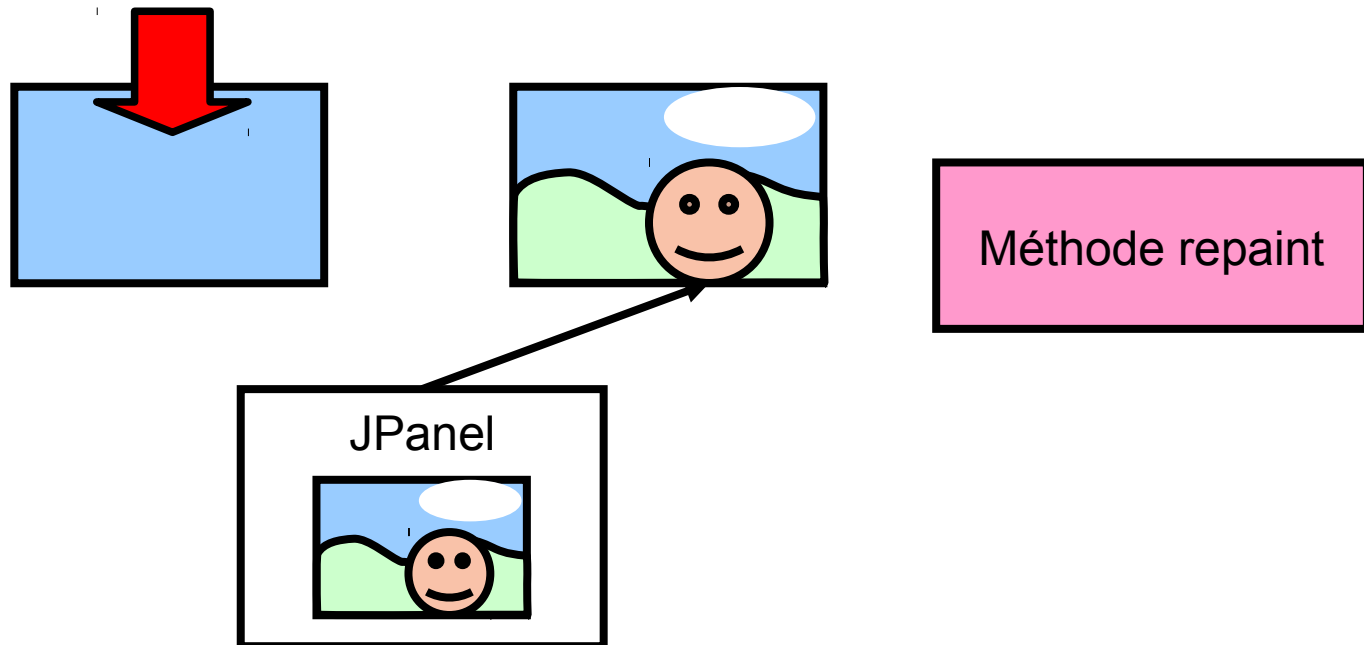
Double buffering

- Principe « Double buffering »
 - Deux images
 - Image affichée et image de travail
 - Repaint inverse les images (copie mémoire rapide)



Double buffering

- Principe « Double buffering »
 - Deux images
 - Image affichée et image de travail
 - Repaint inverse les images (copie mémoire rapide)



Double buffering

- Java
 - Class BufferStrategy
- À partir JFrame
 - Synchronisation verticale
 - Plusieurs buffers

```
// affiche la balle
ObjetBalle b = m.balle;
b.draw(g);

bs.show();
Toolkit.getDefaultToolkit().sync();
g.dispose();
```

Démonstration partie 7

Double buffering

07x03 – Double buffering

- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
 - Vue subjective / changement de repère
 - Double buffering
 - Sprites et animation
 - Scrolling
- Réseau

Animation - sprites

- **Décomposition mouvement**
 - Des mouvements
 - Un sprite pour chaque étape

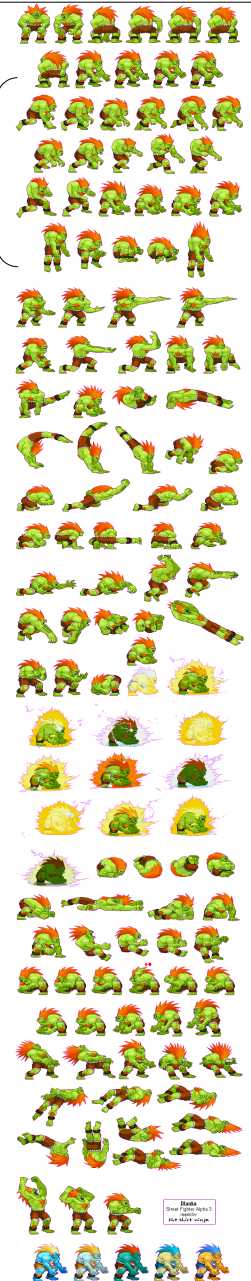
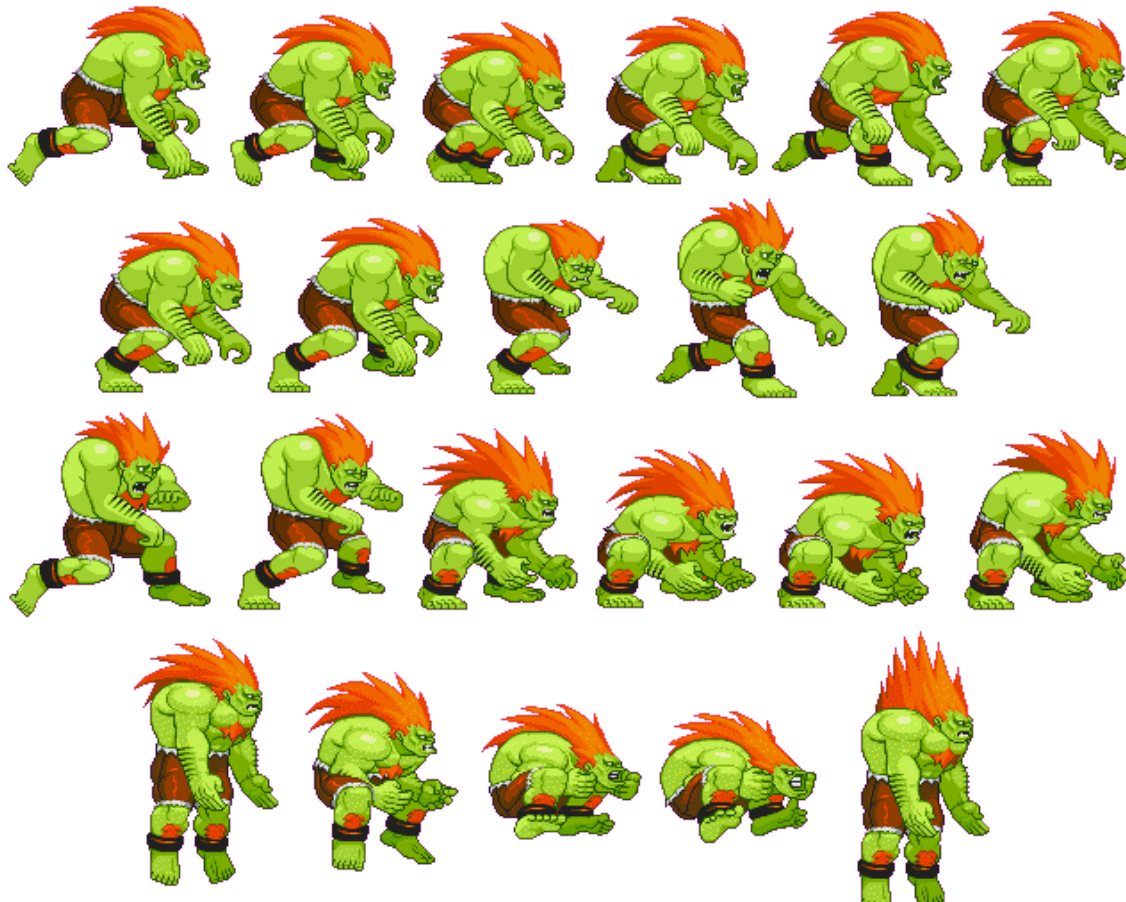
- Décomposition mouvement
 - Des mouvements
 - Un sprite pour chaque étape

SpriteSheet Blanka - SF alpha III



Animation - sprites

- **Décomposition mouvement**
 - Un sprite pour chaque étape



- Utilisation de Sprite
 - Extraction
 - Copie au bon endroit

<http://fivedots.coe.psu.ac.th/~ad/jg/ch04/index.html>

Utilisation de sprites

- Utilisation de Sprite
 - Extraction
 - Copie au bon endroit



Décor (Street fighter 2)

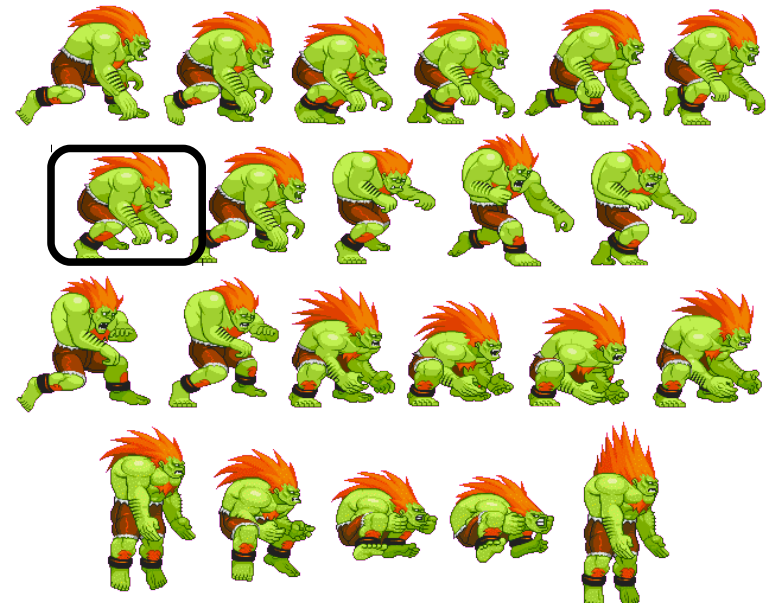
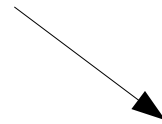
Utilisation de sprites

- Utilisation de Sprite
 - Extraction
 - Copie au bon endroit



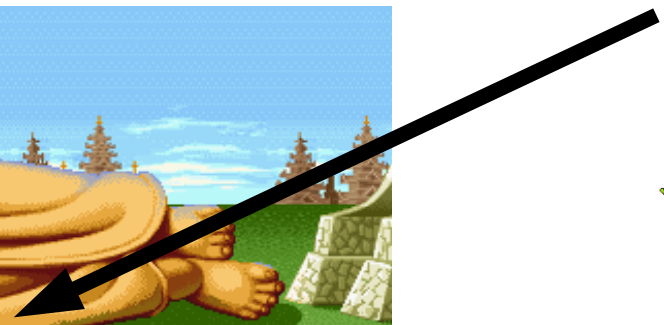
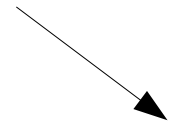
Extraction du décor à afficher

- Choix du sprite



Utilisation de sprites

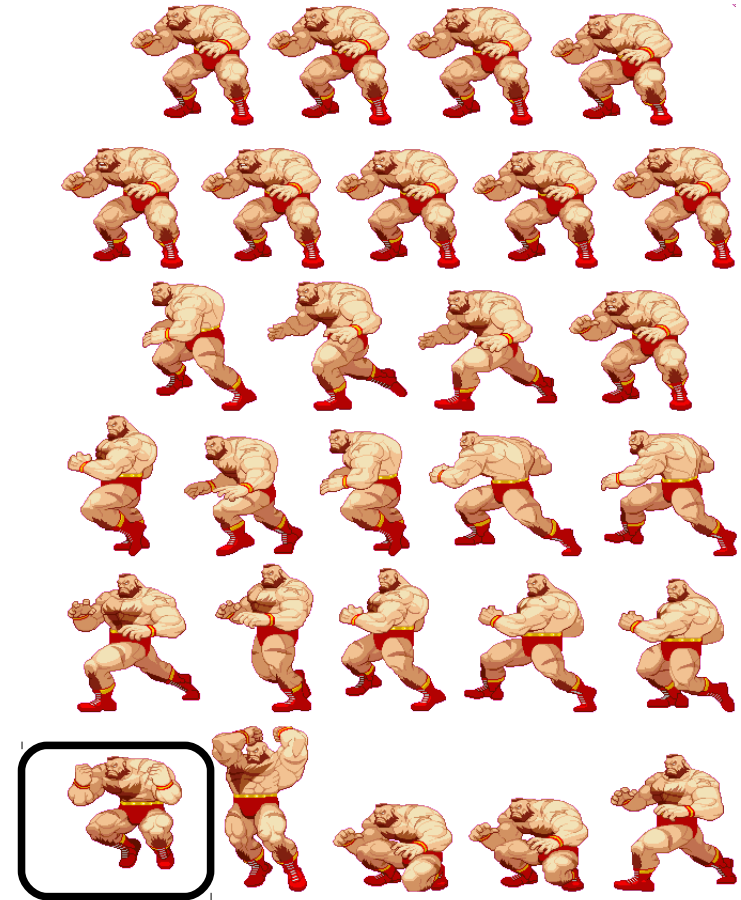
- Copie + gestion transparence



Choix du sprite

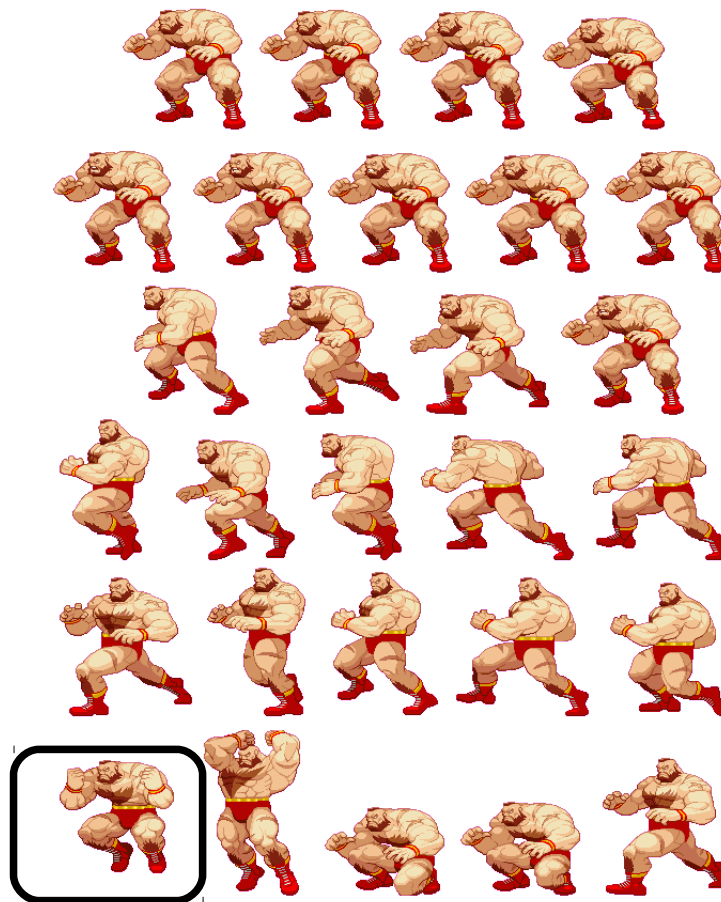
Utilisation de sprites

- Choix sprite (bis)



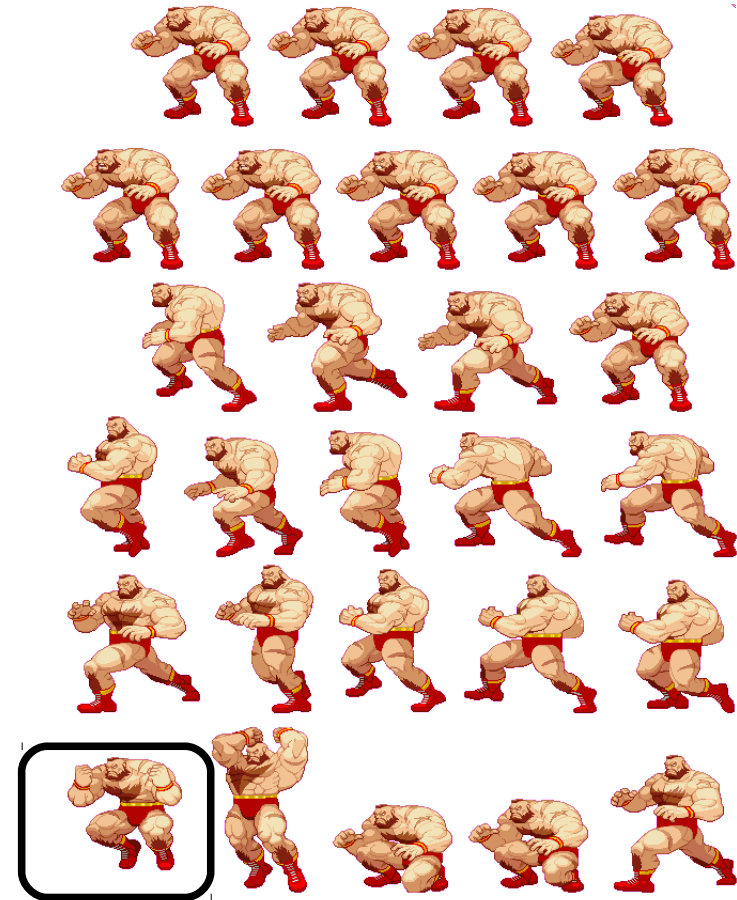
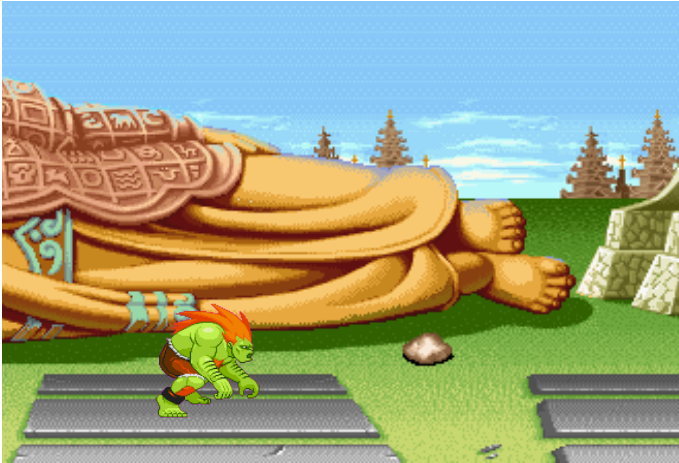
Utilisation de sprites

- Choix sprite (bis)



Utilisation de sprites

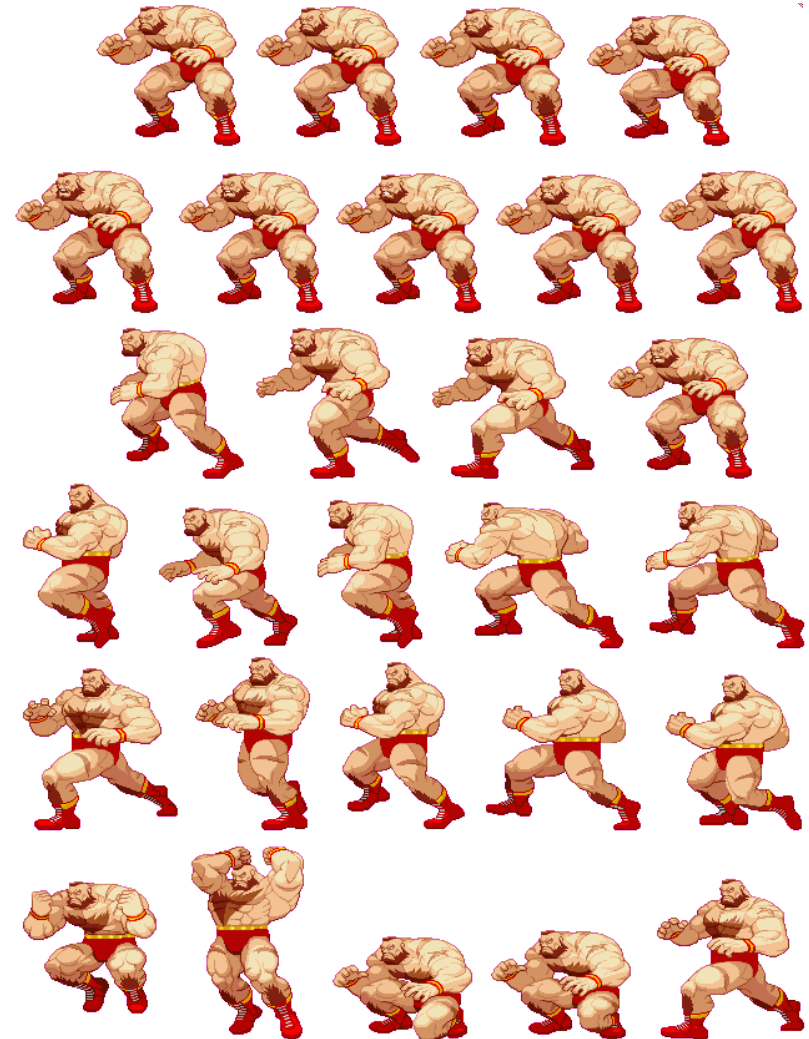
- Choix sprite (bis)



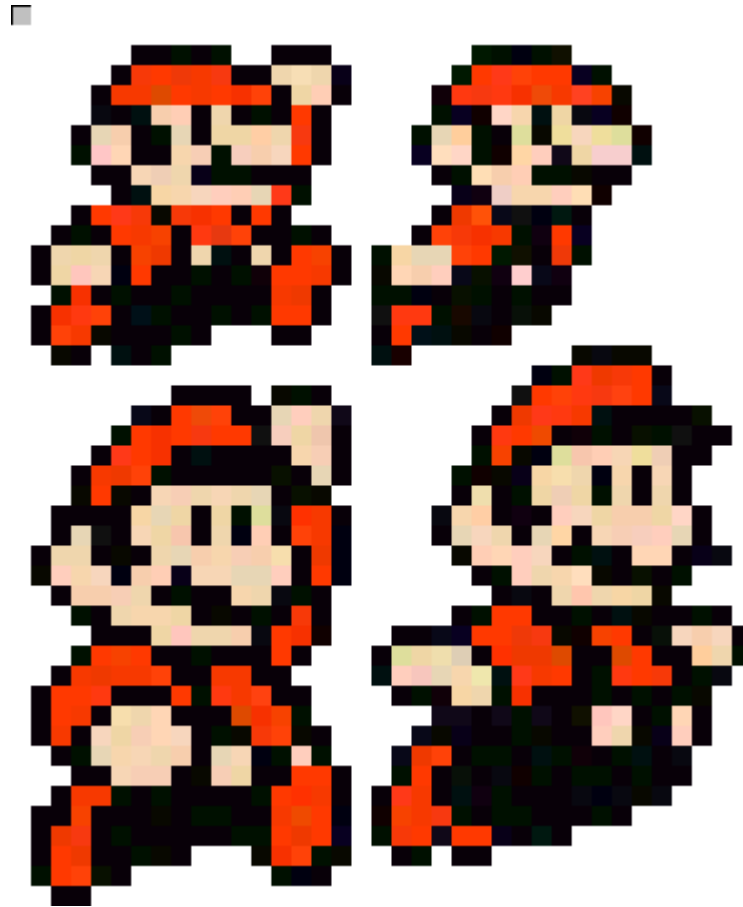
- Image finale



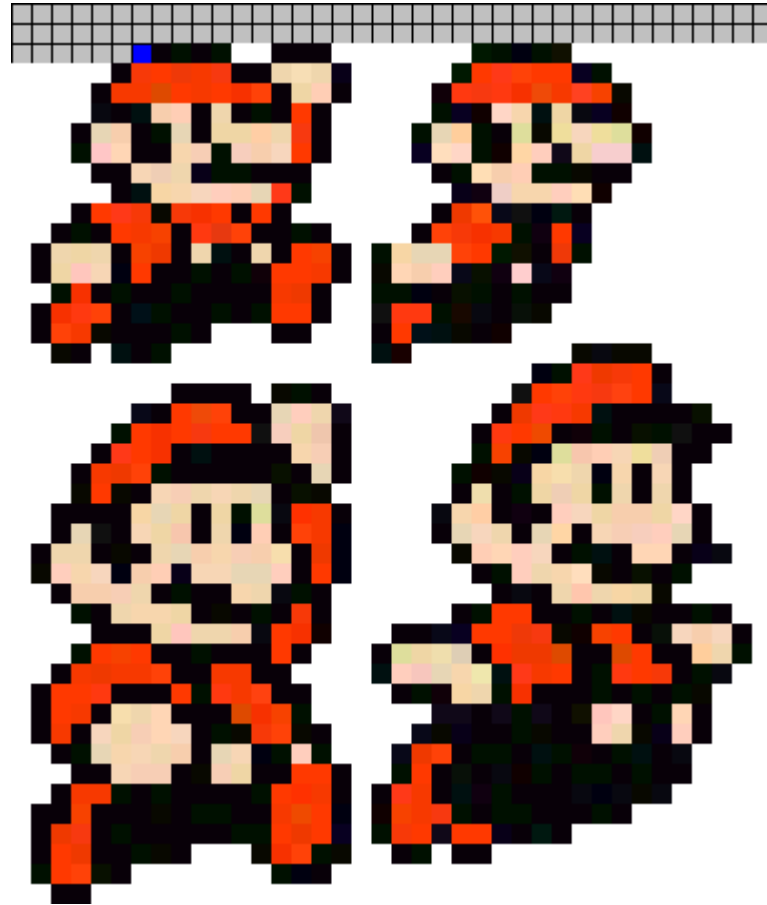
- A la main
 - Définir rectangle
- Automatiquement
 - Zones connexes
 - Sauve fichier



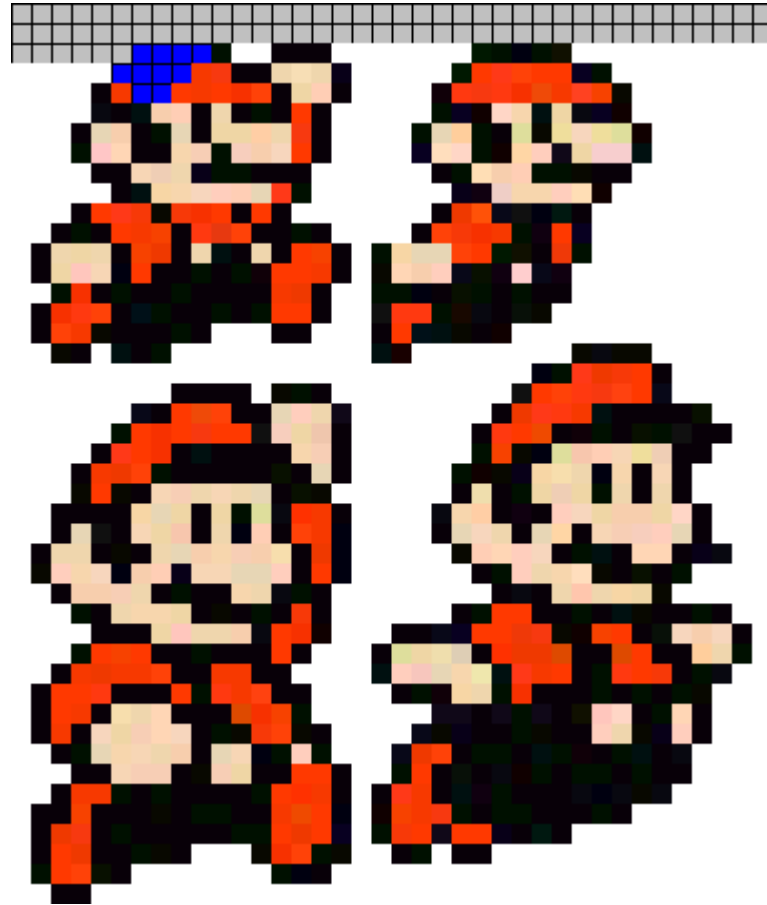
- Partir d'un coin



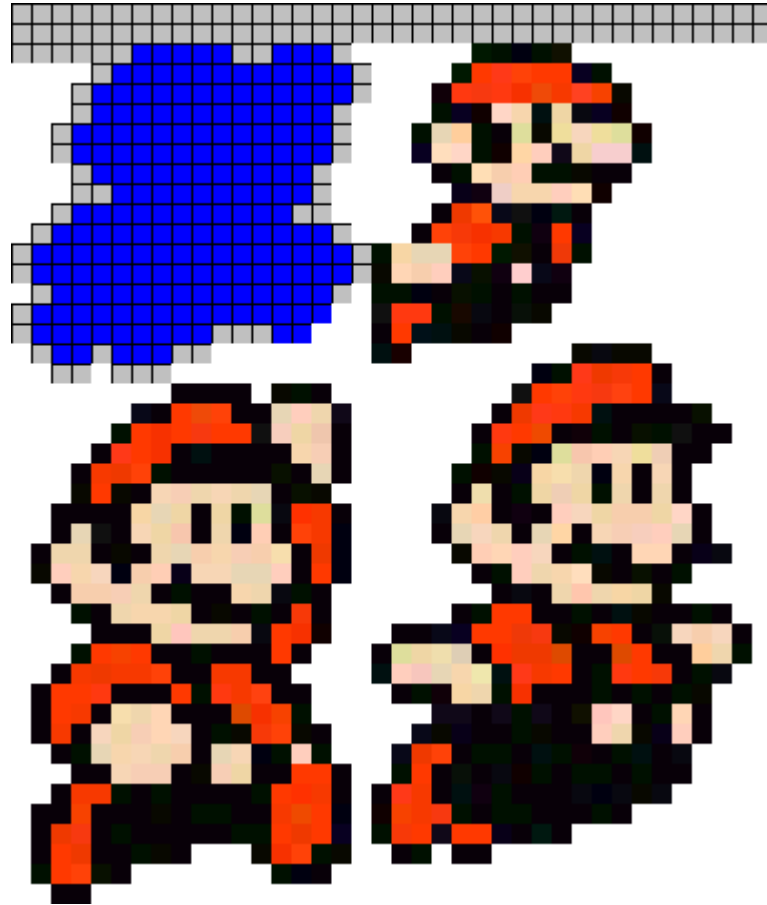
- Partir d'un coin



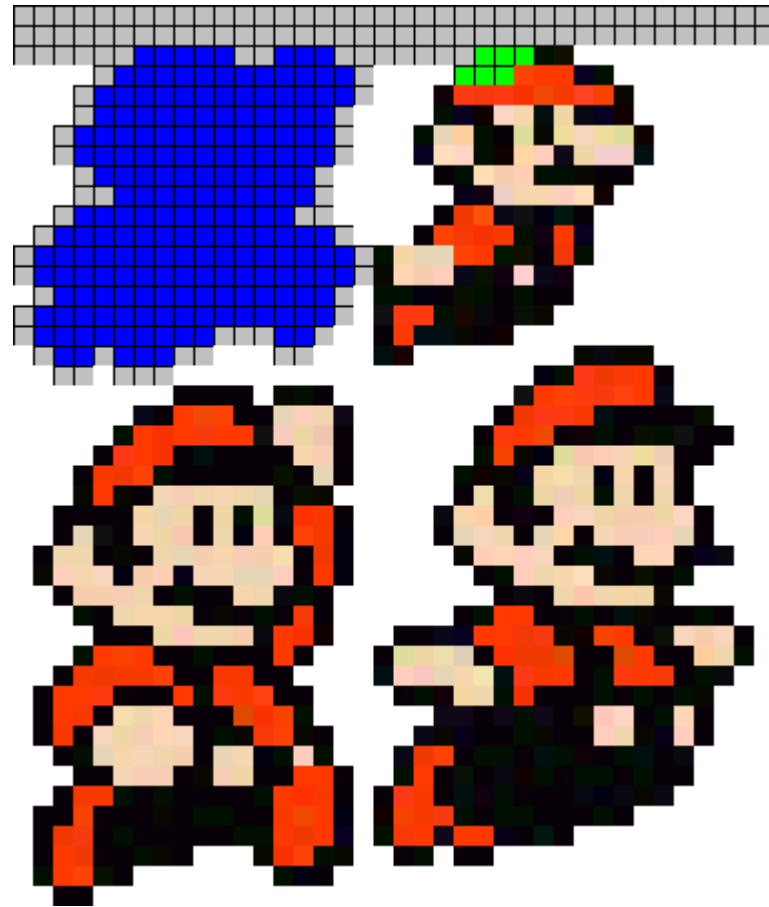
- Partir d'un coin



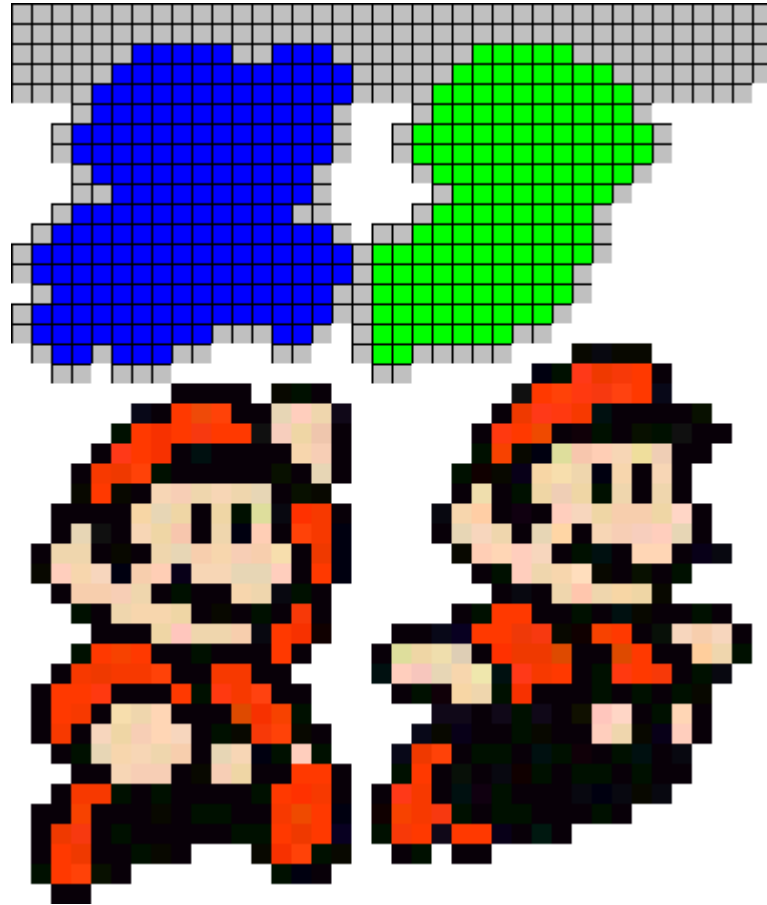
- Partir d'un coin



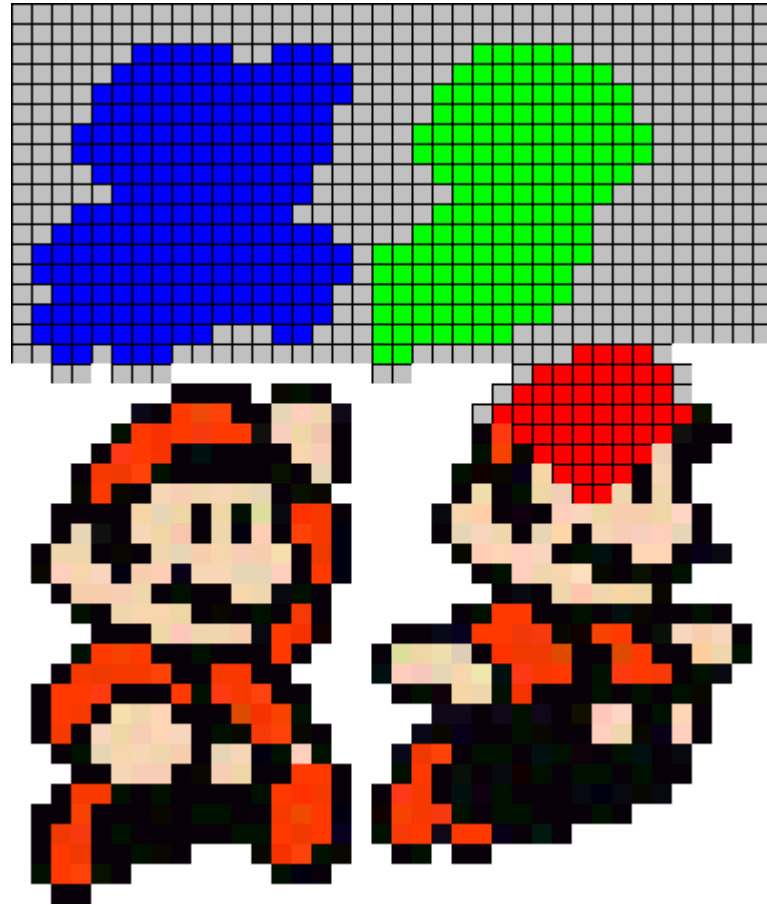
- Partir d'un coin



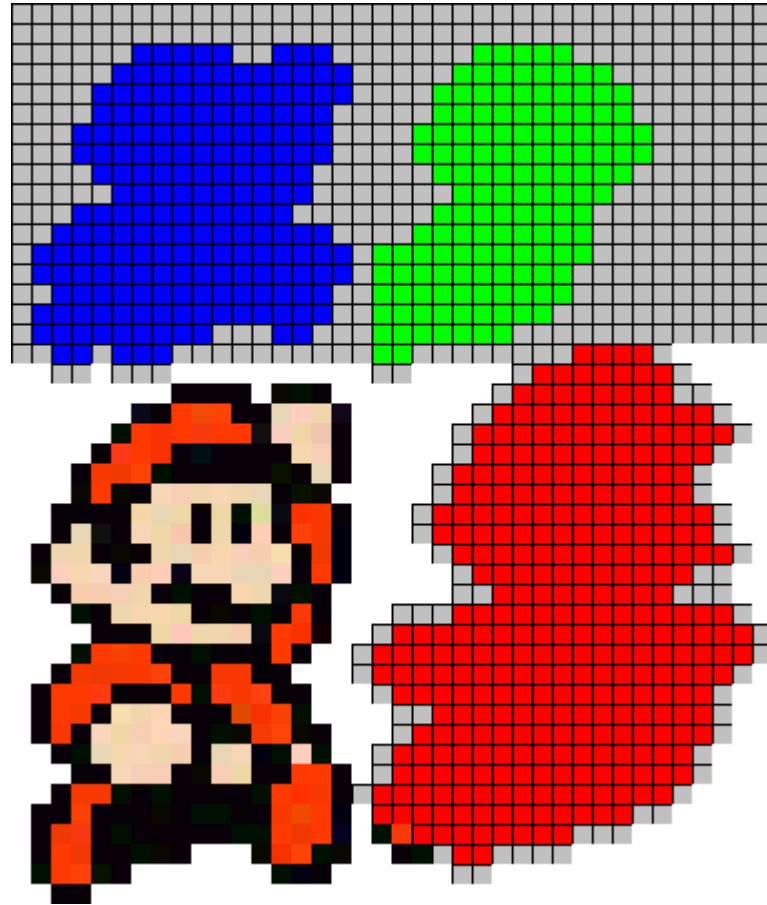
- Partir d'un coin



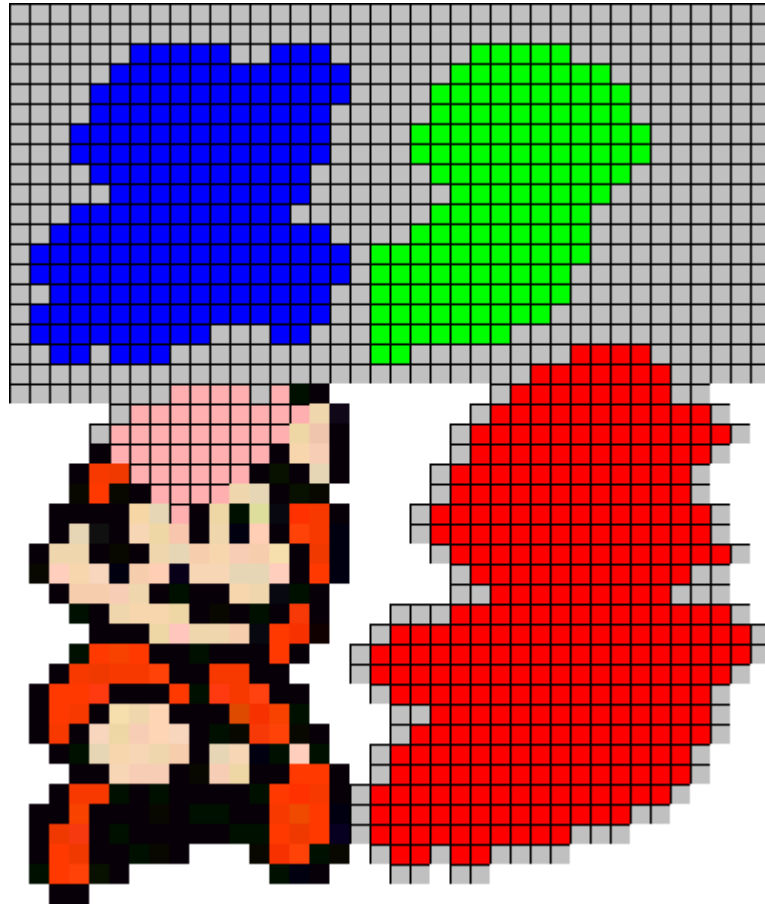
- Partir d'un coin



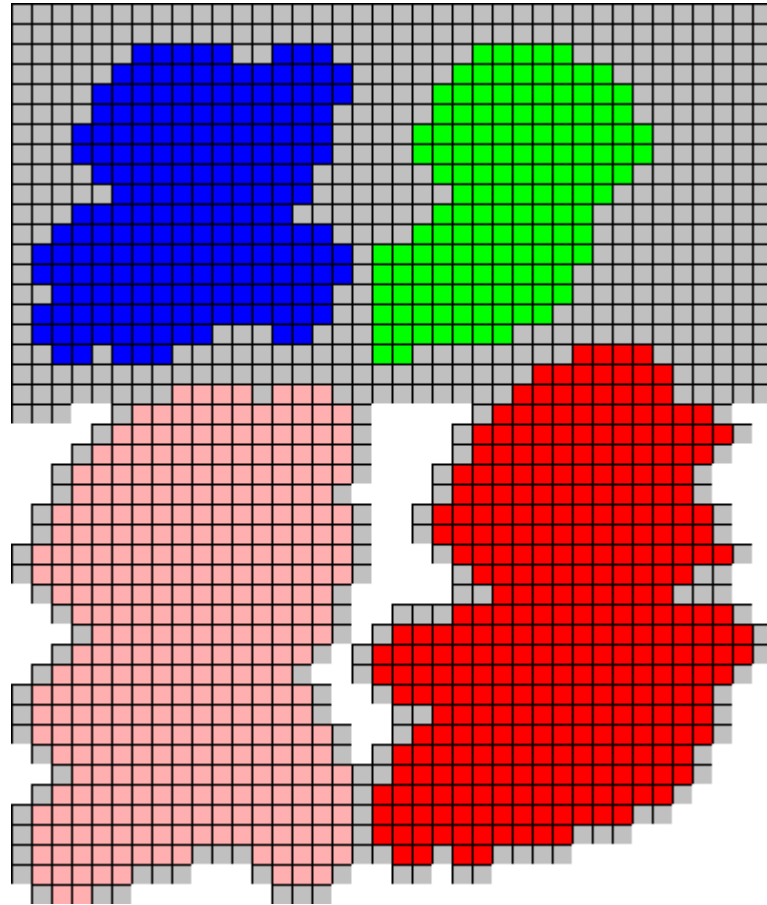
- Partir d'un coin



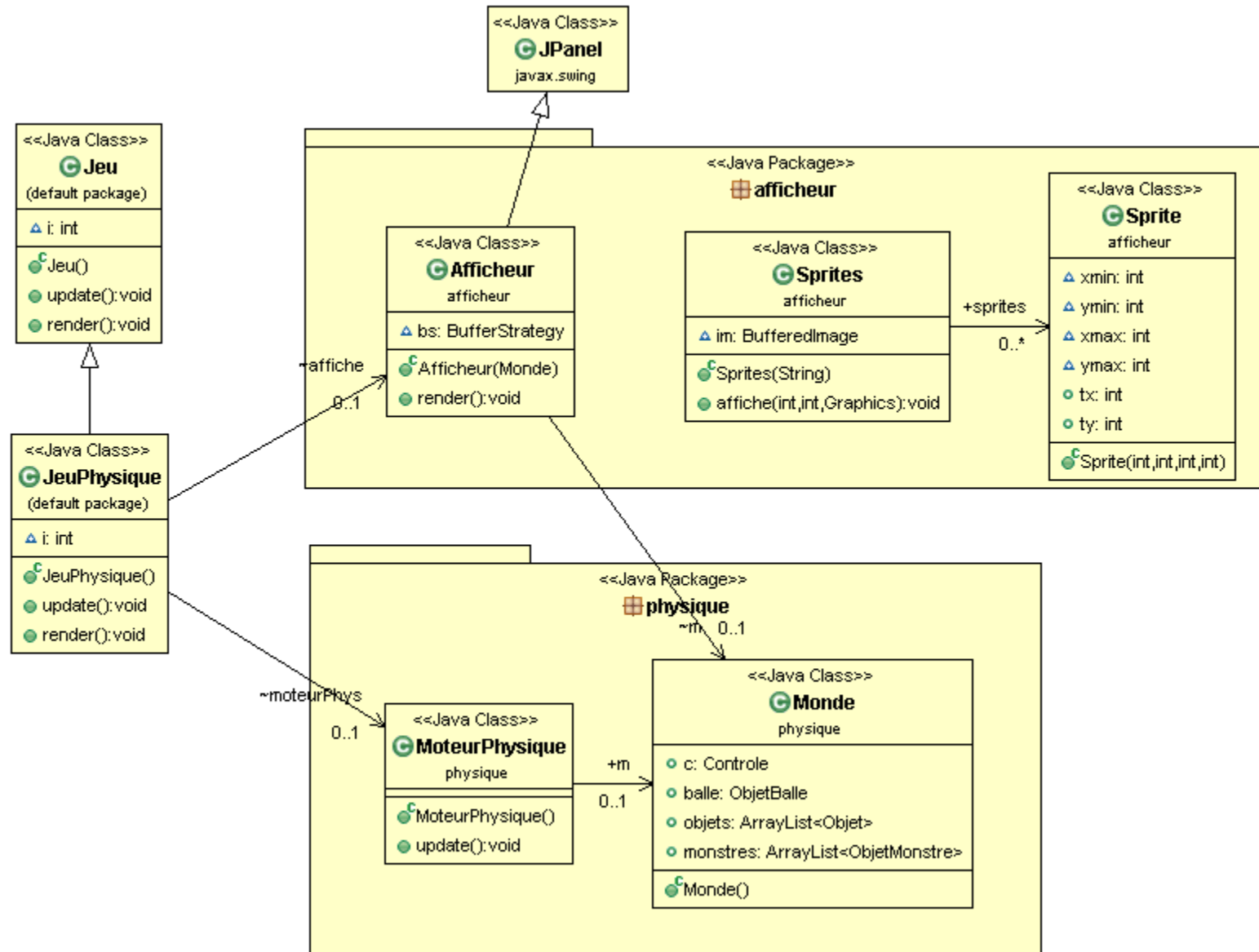
- Partir d'un coin



- Partir d'un coin



Diag classe



Démonstration Partie 8

Ajout sprite

08x01 – extracteur sprite mario

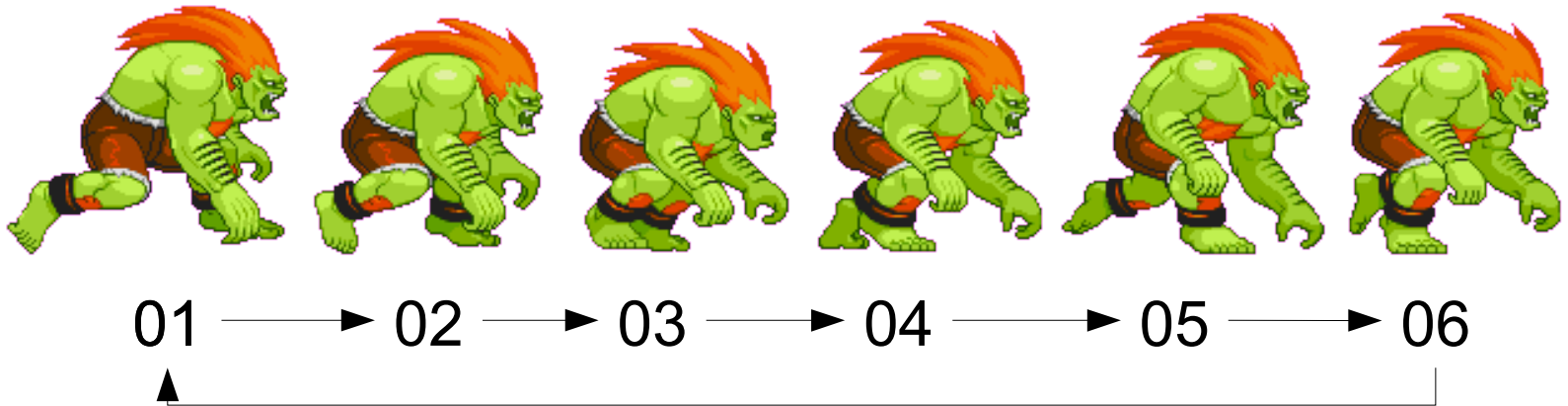
08x02 – extracteur sprite + affiche

08x03 – dessine un sprite + balle

08x04 – dessine un sprite taille adaptée

08x05 – sprite, taille et monde adapté

- Changer sprite fonction du temps



- Pas un par frame
 - Faire un modulo
 - $\text{num} = (\text{it}/10)\%6+1$



09x01 – animation blanka



- Gestionnaire de sprites
- Données
 - SpriteSheet
 - Animations
 - Suite rectangles à extraire
- Comportement
 - Mise à jour des sprites



- **Marche**

- 1 ==> Rect (100,450,20,30)
- 2 ==> Rect (140,450,20,30)
- ...

- **Attaque poing**

- 1 ==> Rect (100,500,20,30)
- 2 ==> Rect (140,500,30,40)
- ...

- **Attaque Pied**

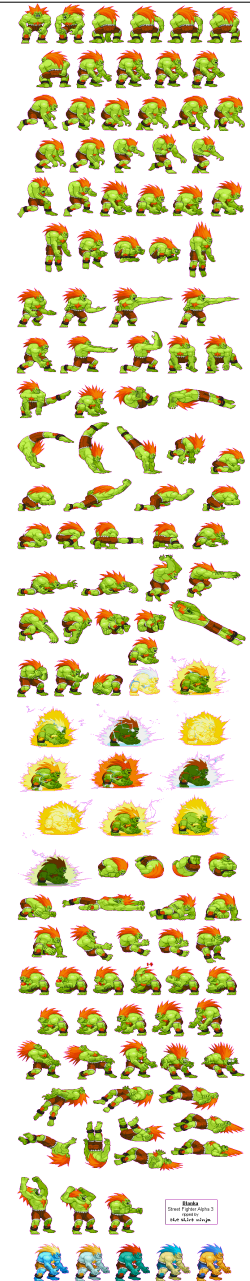
- ...

+



Animation (exemple)

Int numero
String comportement



File:
D:\...
...
...

Animation (exemple)

C
o
m
p
o
r
t
e
m
e
n
t

Int numero
String comportement

Marche



fonction numéro animation 1 → 2 → 3 → 4 → 5 → 6 → 1

Poing



fonction numéro animation 1 → 2 → 2 → 3 → 3

Pied



fonction numéro animation 1 → 1 → 2 → 2 → 3 → 3




Bank
© 2005
All rights reserved


Animation (exemple)

C
o
m
p
o
r
t
e
m
e
n
t


Int numero
String comportement

Marche 

fonction numéro animation 1 → 2 → 3 → 4 → 5 → 6 → 1

Poing 

fonction numéro animation 1 → 2 → 2 → 3 → 3

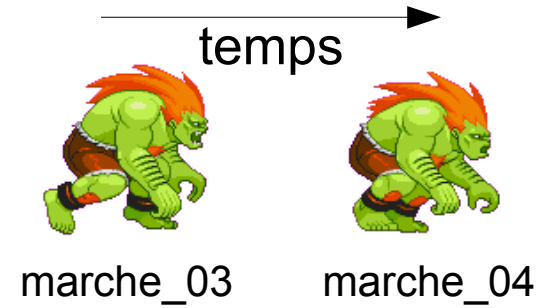
Pied 

fonction numéro animation 1 → 1 → 2 → 2 → 3 → 3

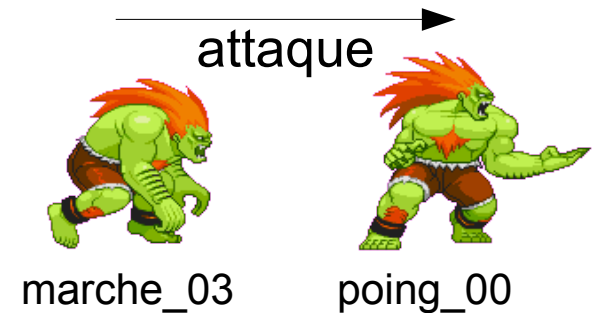
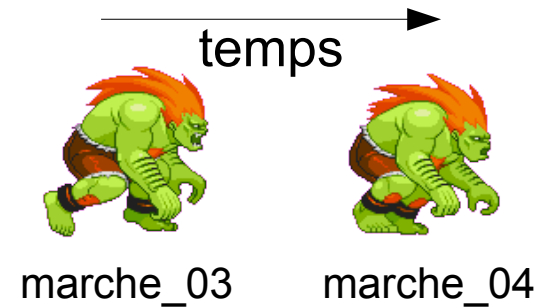


Bank
© 2000
All rights reserved

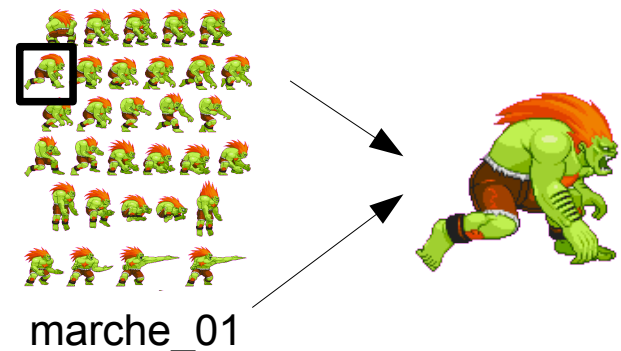
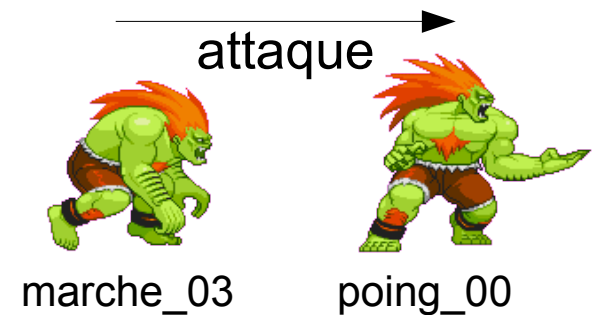
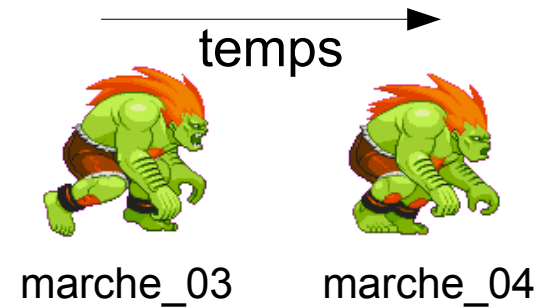
- **anime()**
 - Modifie numéro de frame
 - Gère temps et cycles



- **anime()**
 - Modifie numéro de frame
 - Gère temps et cycles
- **changeCpt(int cpt)**
 - Change comportement
 - Initialise numéro



- **anime()**
 - Modifie numéro de frame
 - Gère temps et cycles
- **changeCpt(int cpt)**
 - Change comportement
 - Initialise numéro
- **afficheSprite(int x, int y)**
 - Comportement + numéro
 - Retourne image extraite



- **anime()**
 - Modifie numéro de frame
 - Gère temps et cycles
- **changeCpt(int cpt)**
 - Change comportement
 - Initialise numéro
- **afficheSprite(int x, int y)**
 - Comportement + numéro
 - Retourne image extraite



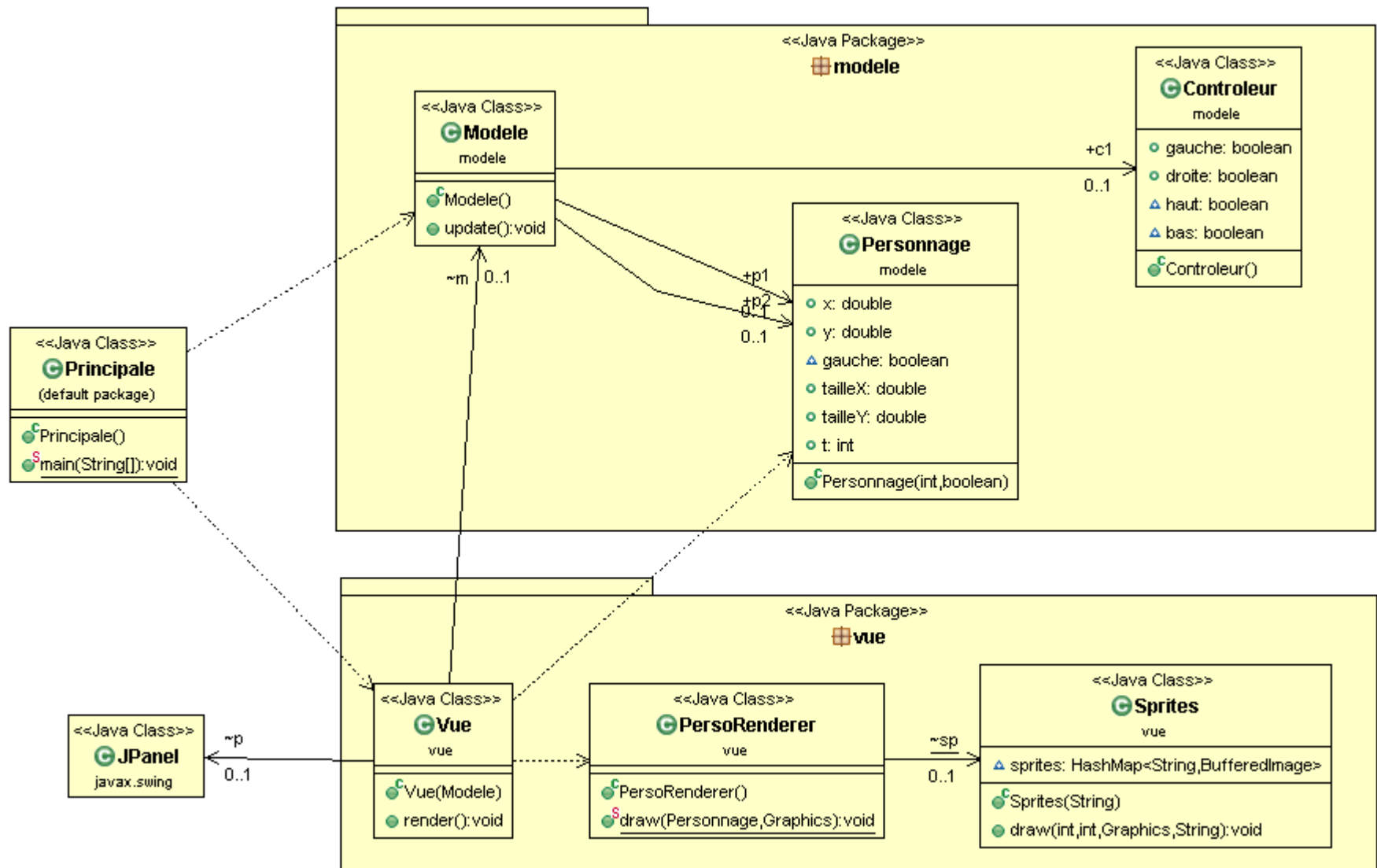
Encapsule
l'animation

- **anime()**
 - Modifie numéro de frame
 - Gère temps et cycles
- **changeCpt(int cpt)**
 - Change comportement
 - Initialise numéro
- **afficheSprite(int x, int y)**
 - Comportement + numéro
 - Retourne image extraite

Encapsule
l'animation

Memes
comportements
que IA / jeu

Animation blanka



Démonstration 09

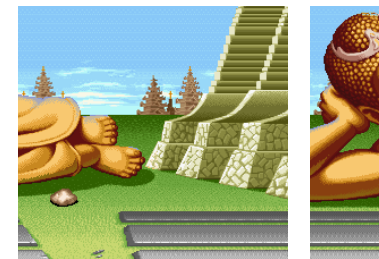
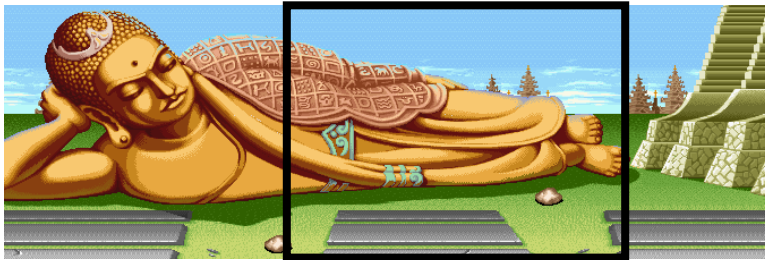
Animation de sprite

- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
 - Vue subjective / changement de repère
 - Double buffering
 - Sprites et animation
 - Scrolling
- Réseau

Quelques exemples

- **Scrolling**

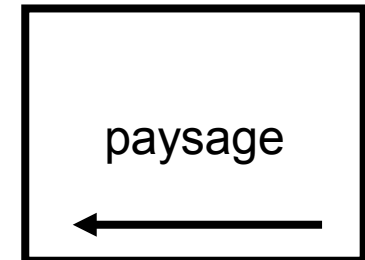
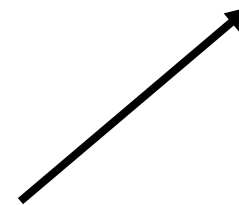
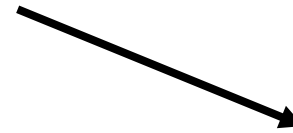
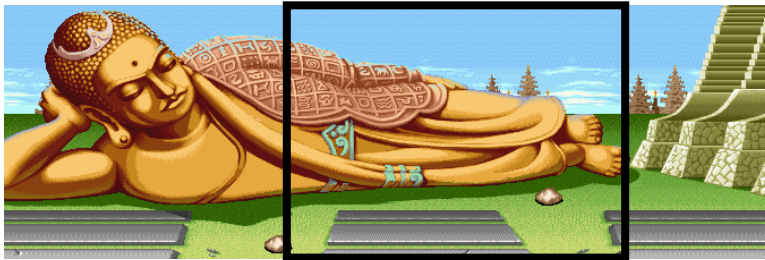
- Défiler un décor copie morceaux image



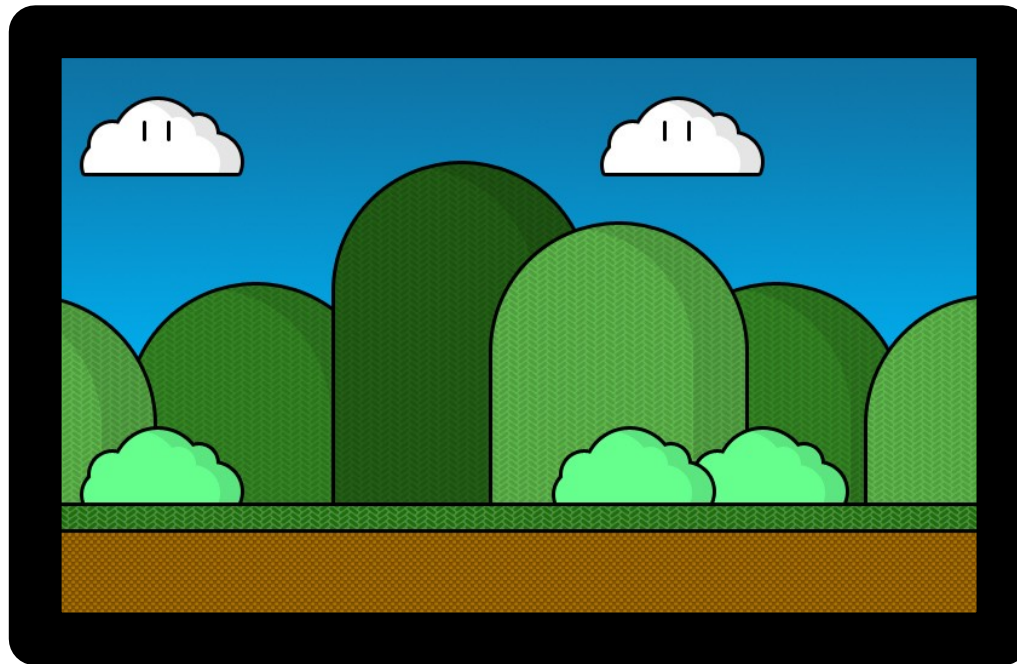
Quelques exemples

- **Scrolling**

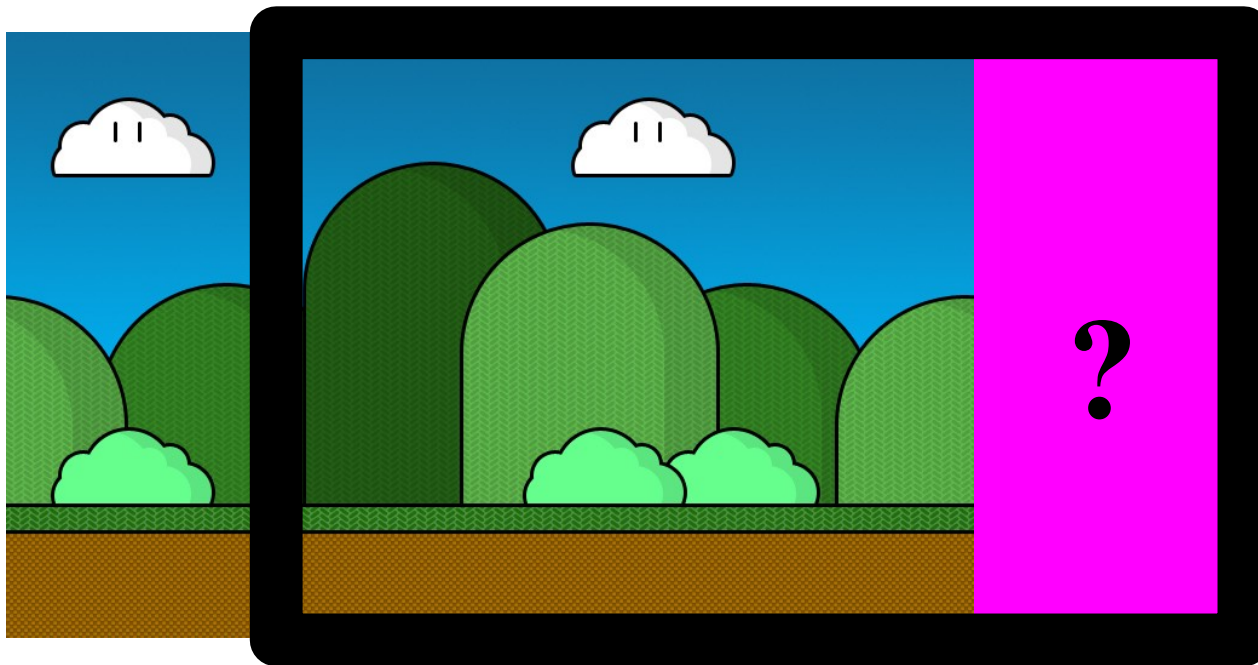
- Défiler un décor copie morceaux image



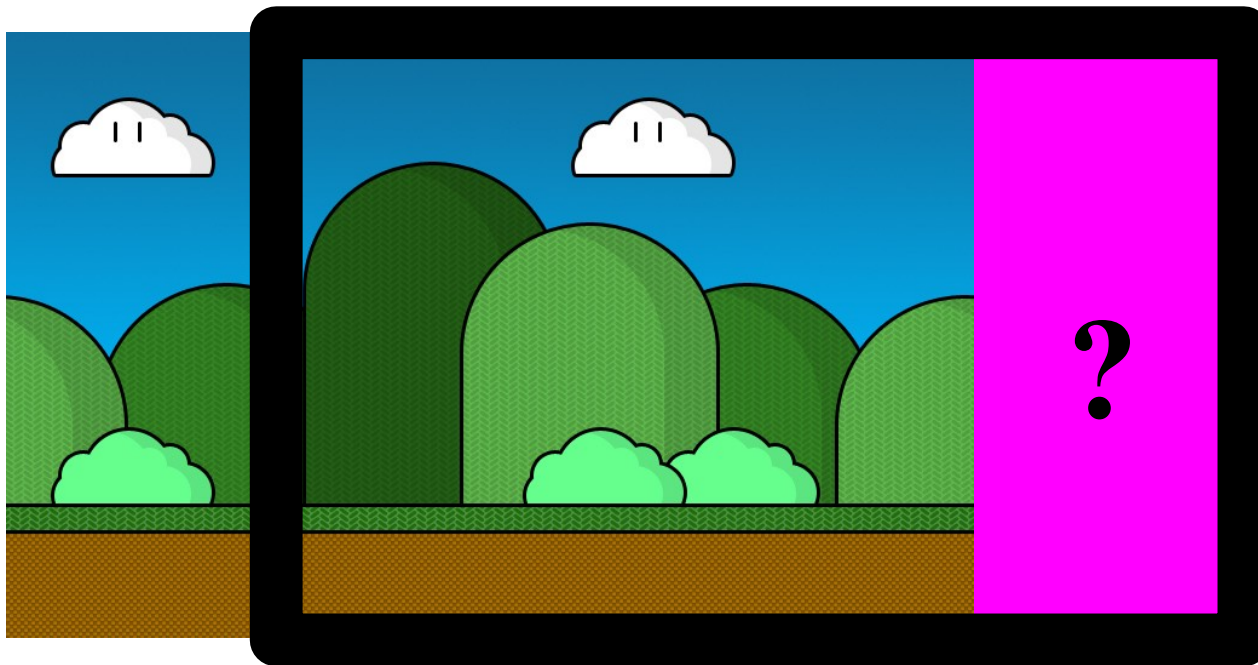
- Toute l'image est affichée



- Décalage de 1 pixel

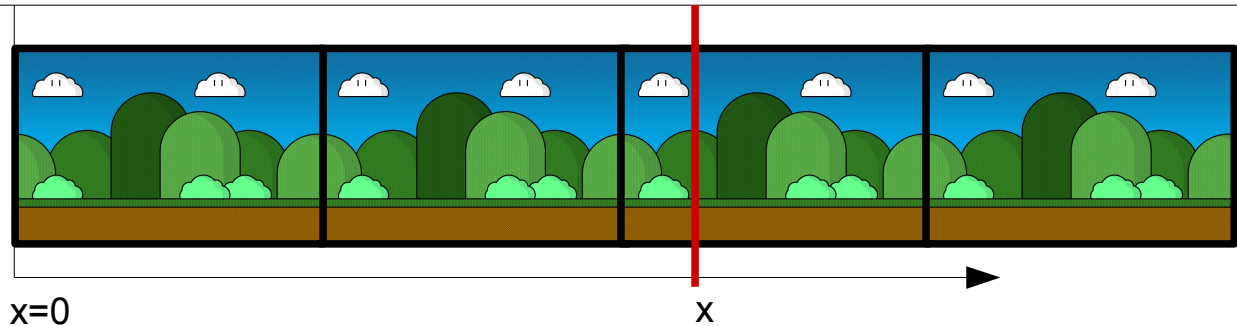


- Décalage de 1 pixel



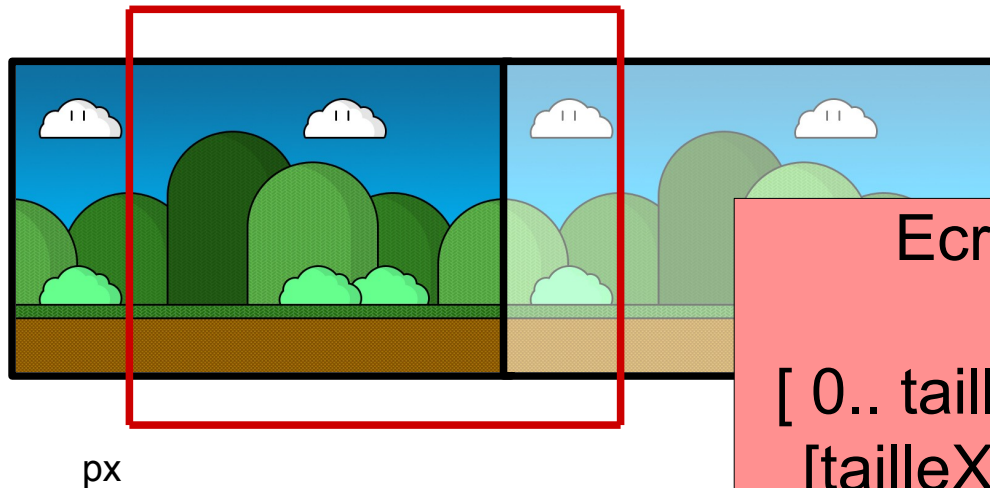
Code Scrolling

- Scrolling



- Position de x dans image n

- $Px = (x \% \text{tailleX})$

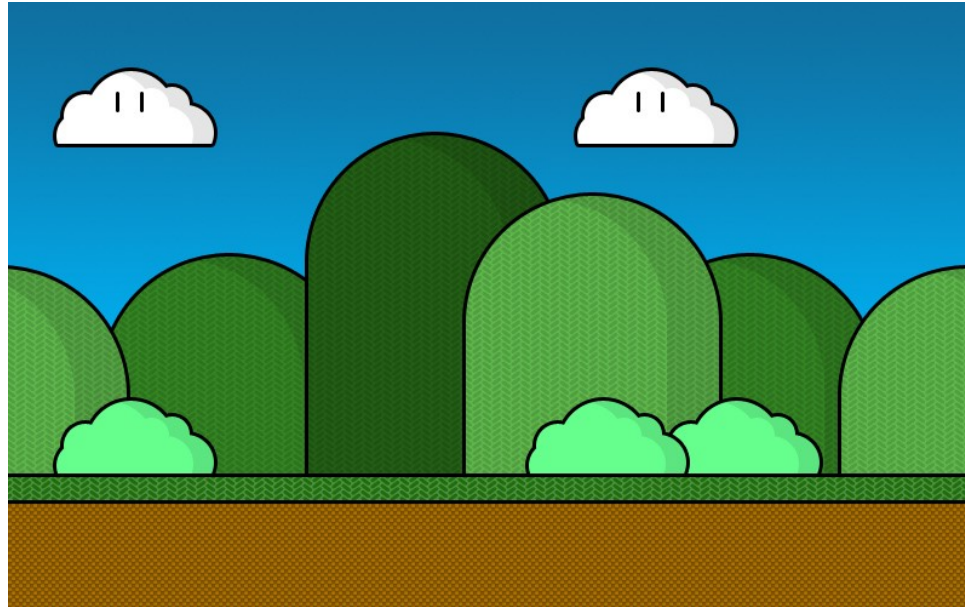


Ecran => image source

$[0.. \text{tailleX}-px] \Rightarrow [px .. \text{tailleX}]$
 $[\text{tailleX}-px .. \text{tailleX}] \Rightarrow [0 .. px]$

Code Scrolling

- Scrolling



```
public void affiche(int x,Graphics g)
{
    //on se ramene au repère du plan
    x=(x%wx);

    //on affiche sur l'ecran 0 ==> wx-x image source de x à wx
    g.drawImage(im, 0 ,0 , wx-x, wy, x, 0, wx, wy,null);

    //on affiche sur l'ecran wx-x ==> wx image source de 0 à x
    g.drawImage(im, wx-x ,0 , wx, wy, 0, 0, x, wy,null);
}
```

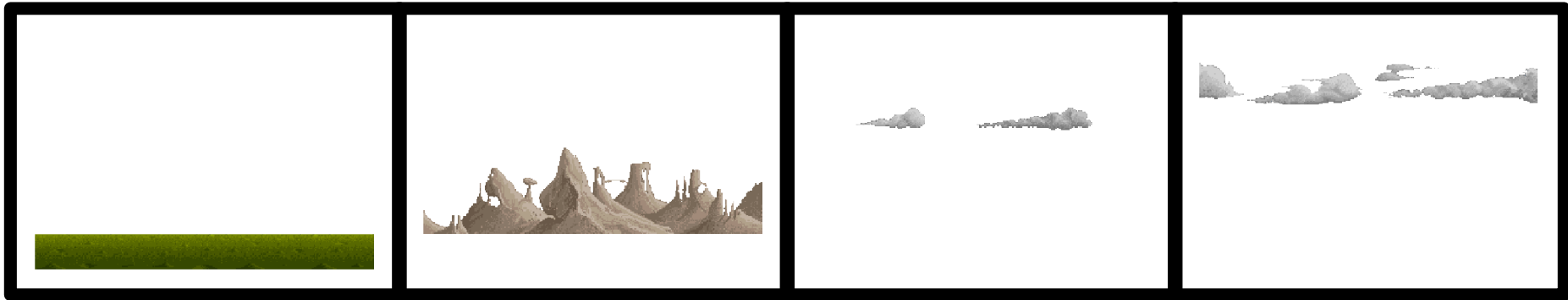
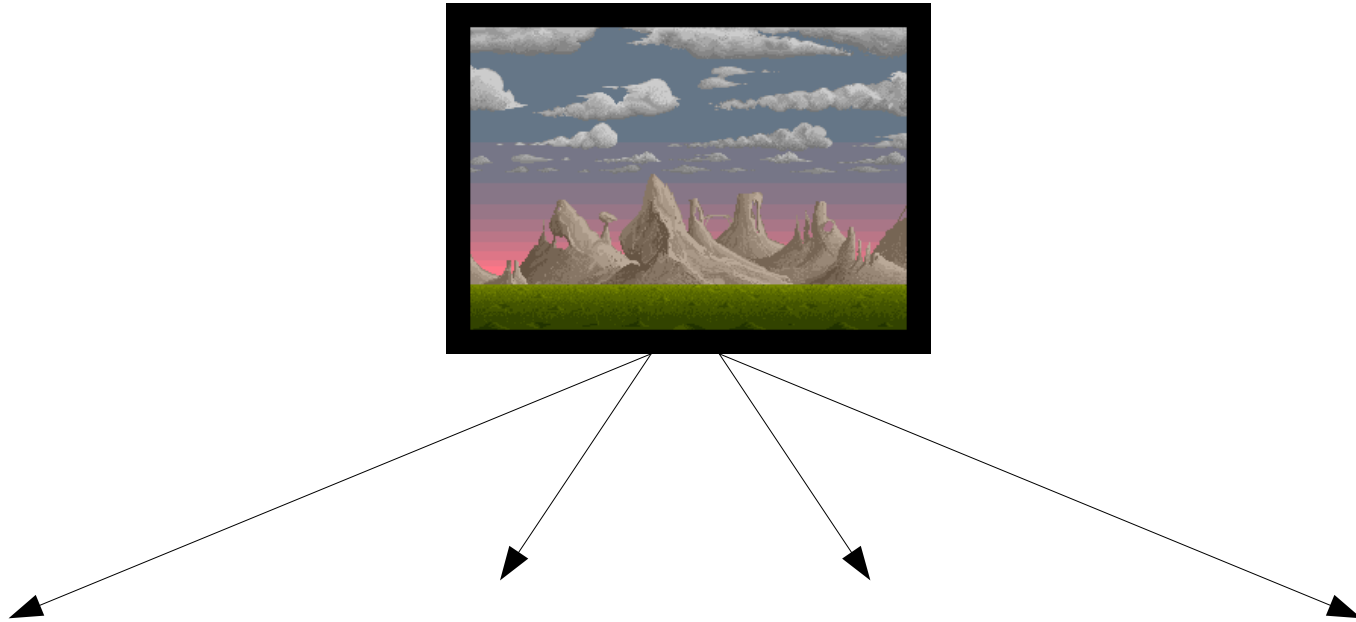
Démonstration

Scrolling

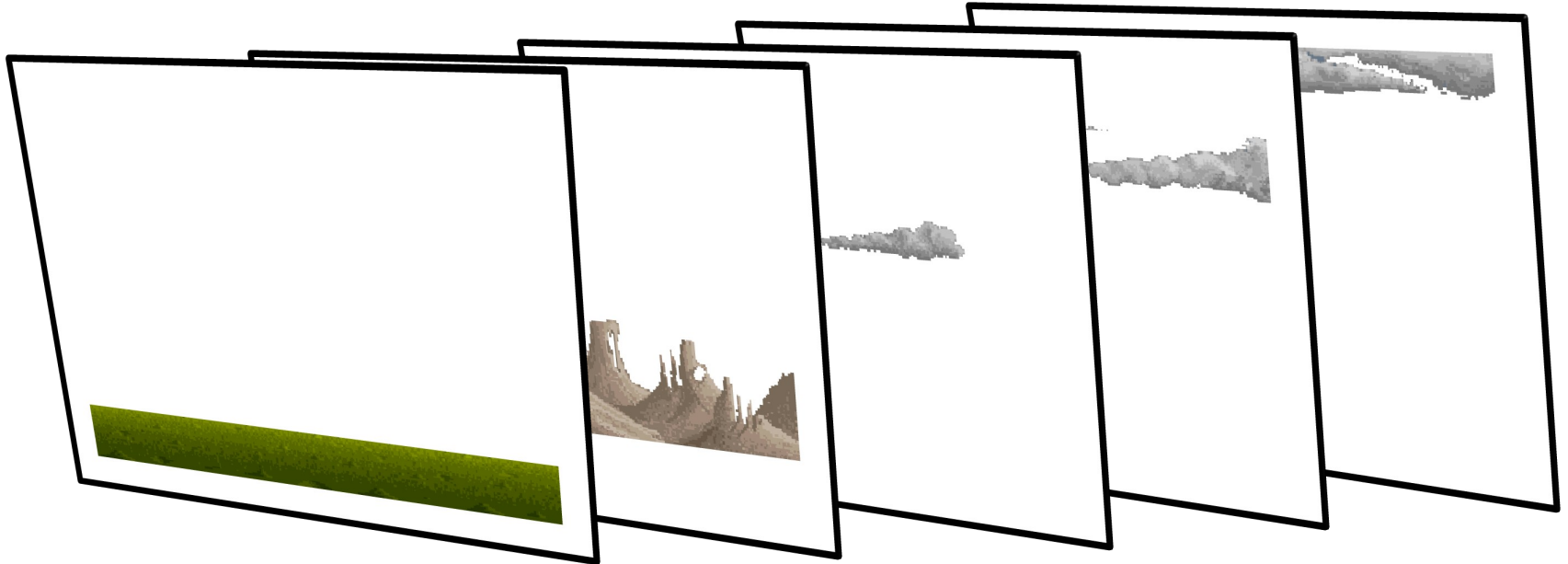
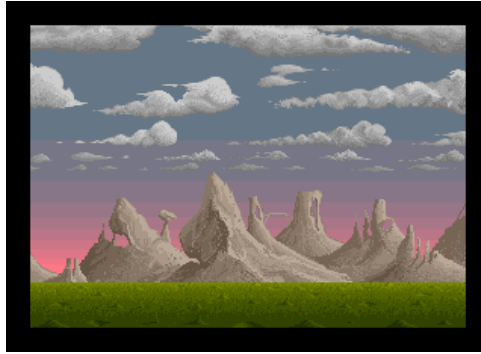
- **Shadow of the beast (1989)**
 - Scrolling à 13 plans



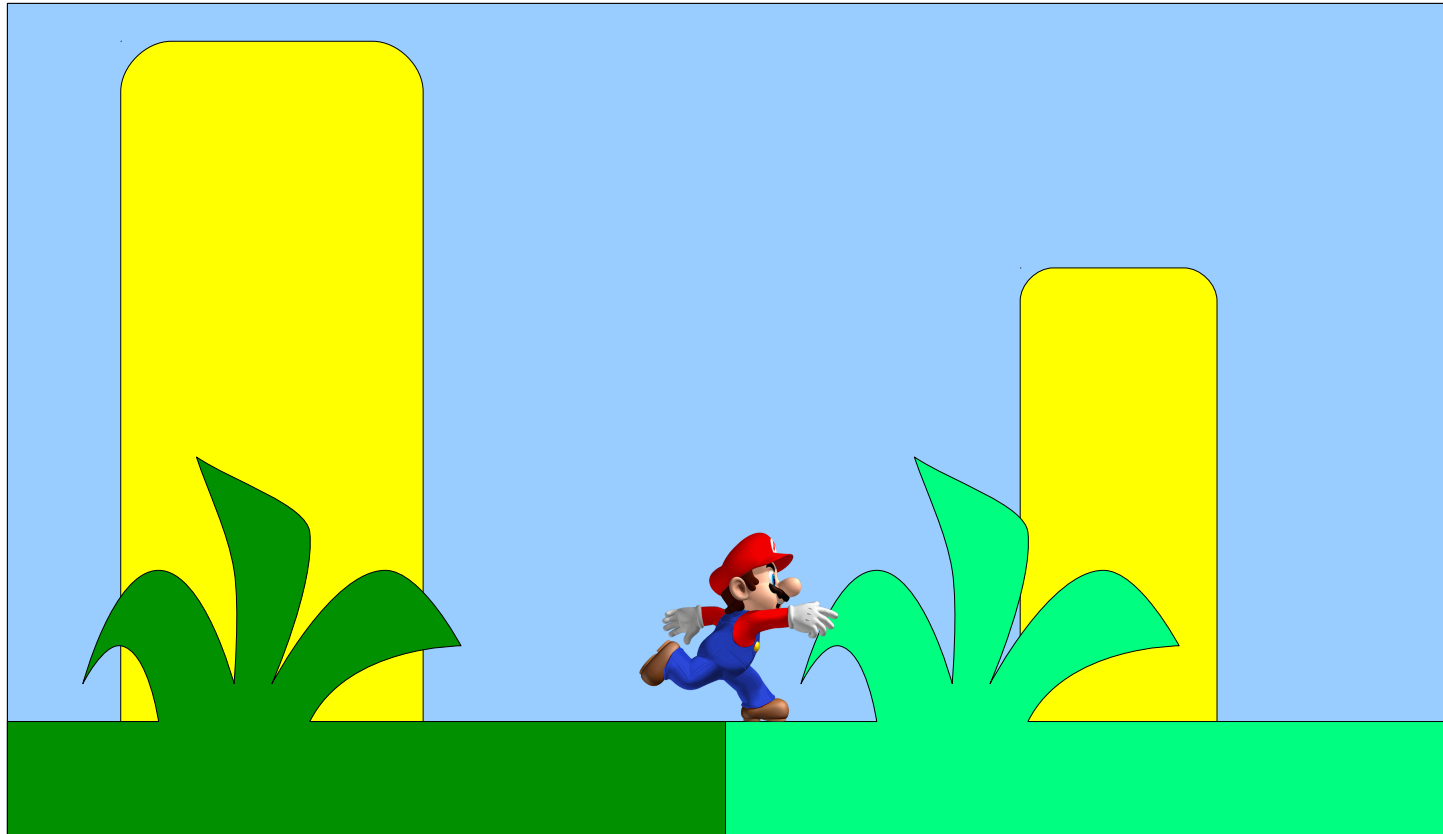
Décomposition en plans



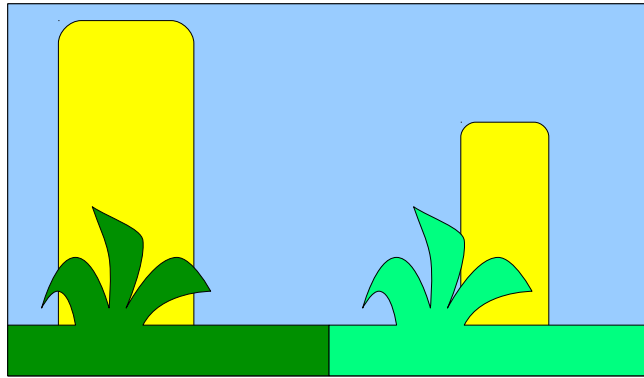
Décomposition en plans



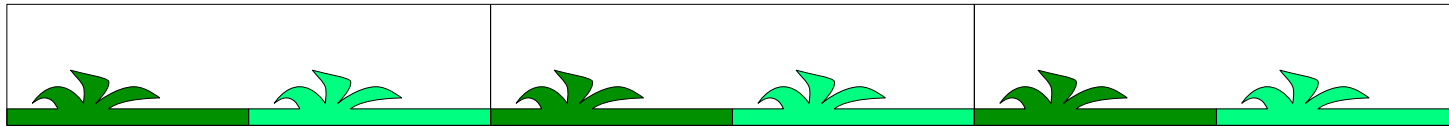
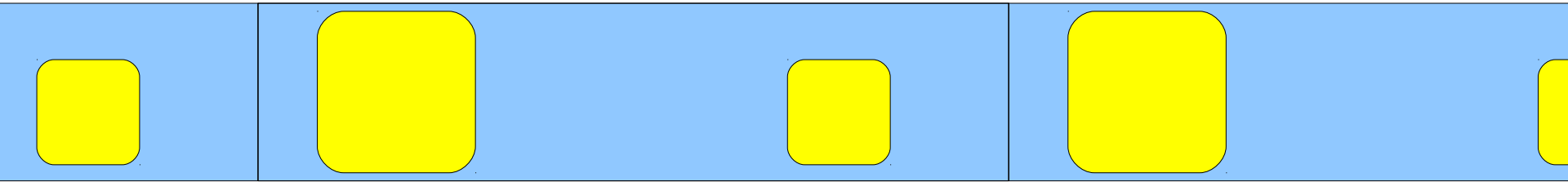
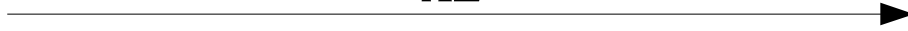
Défilement parallaxe



Défilement parallaxe



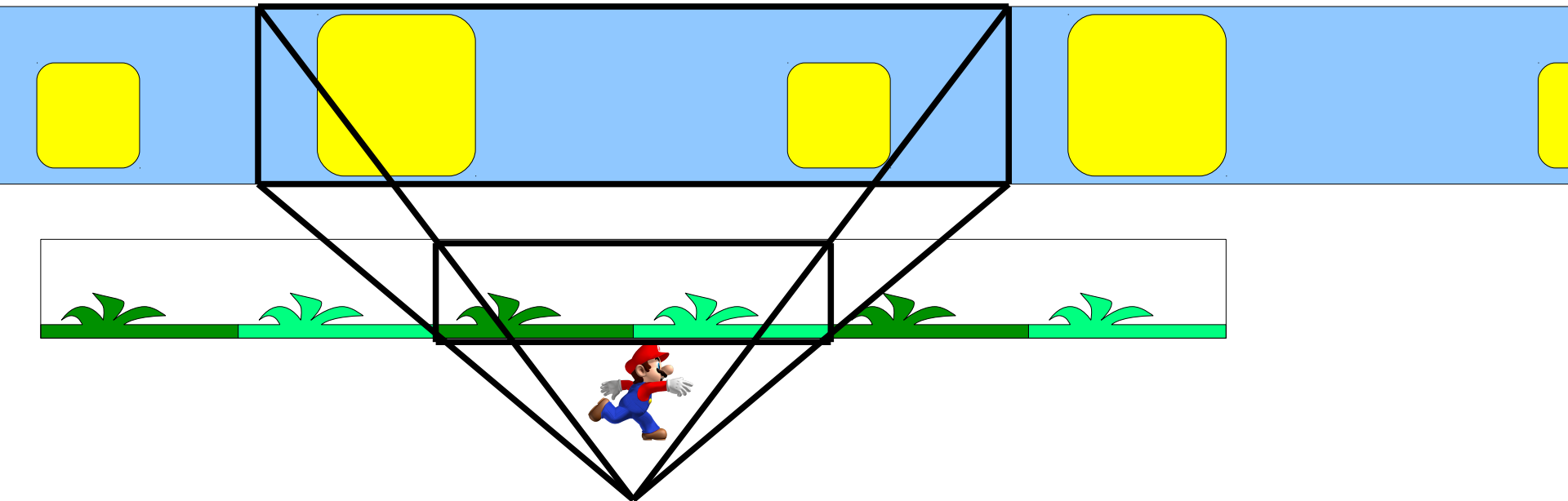
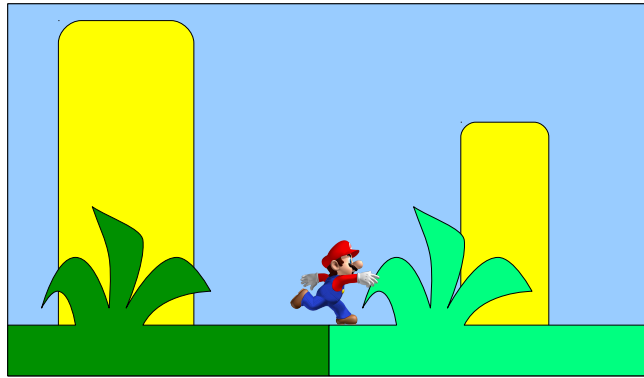
x2



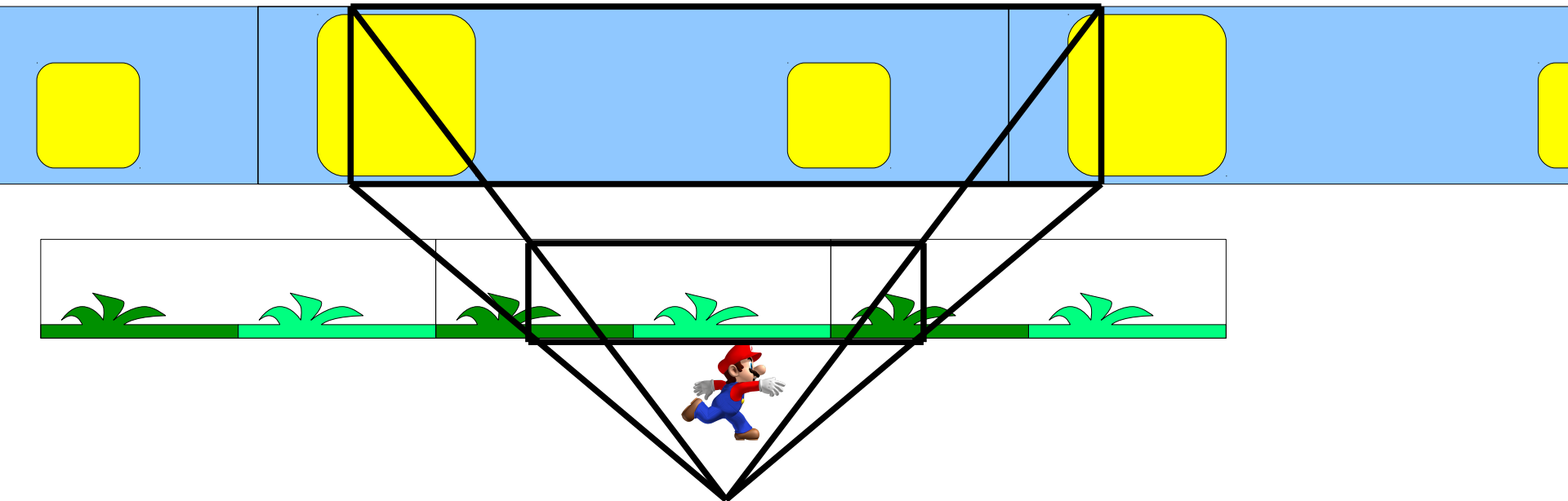
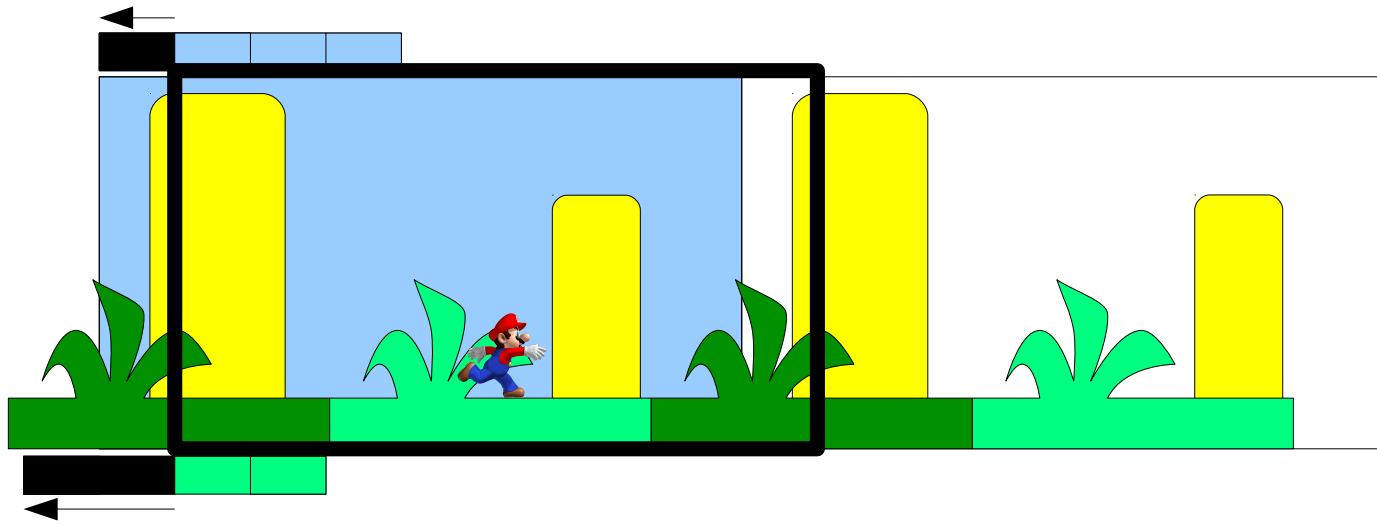
x1



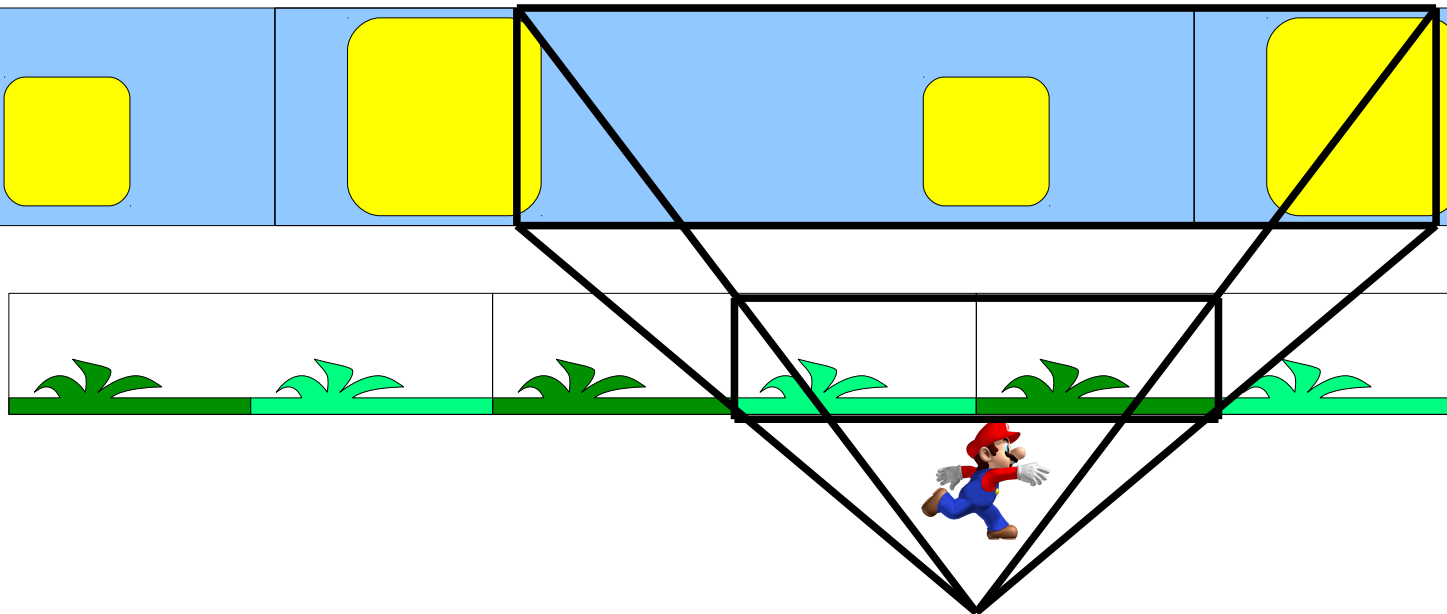
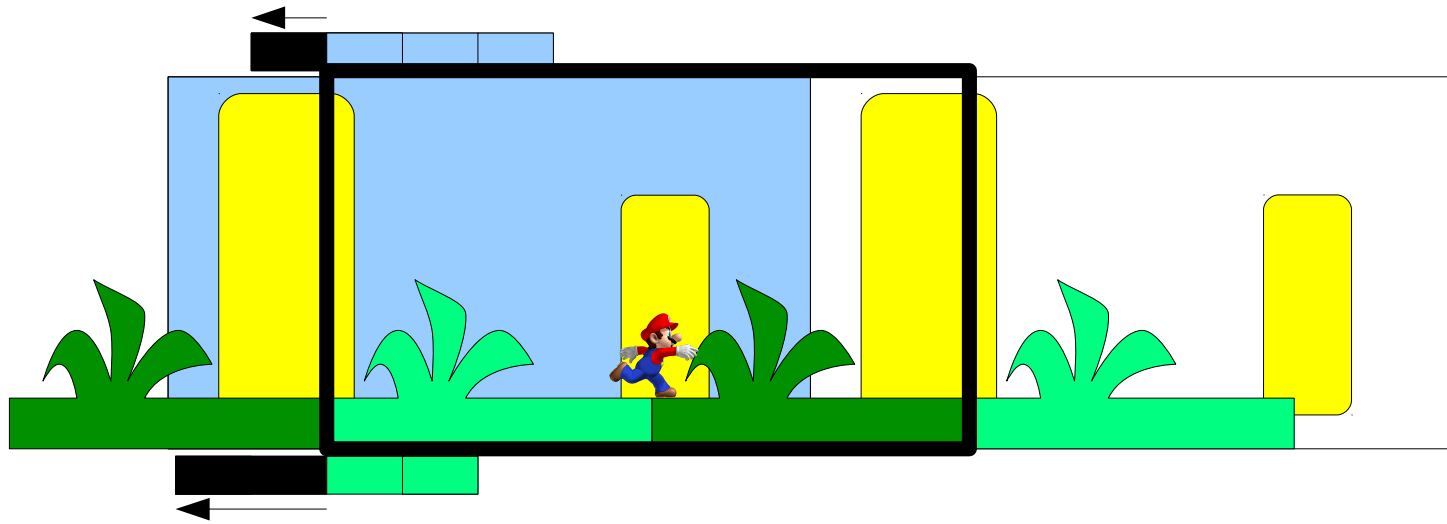
Défilement parallaxe



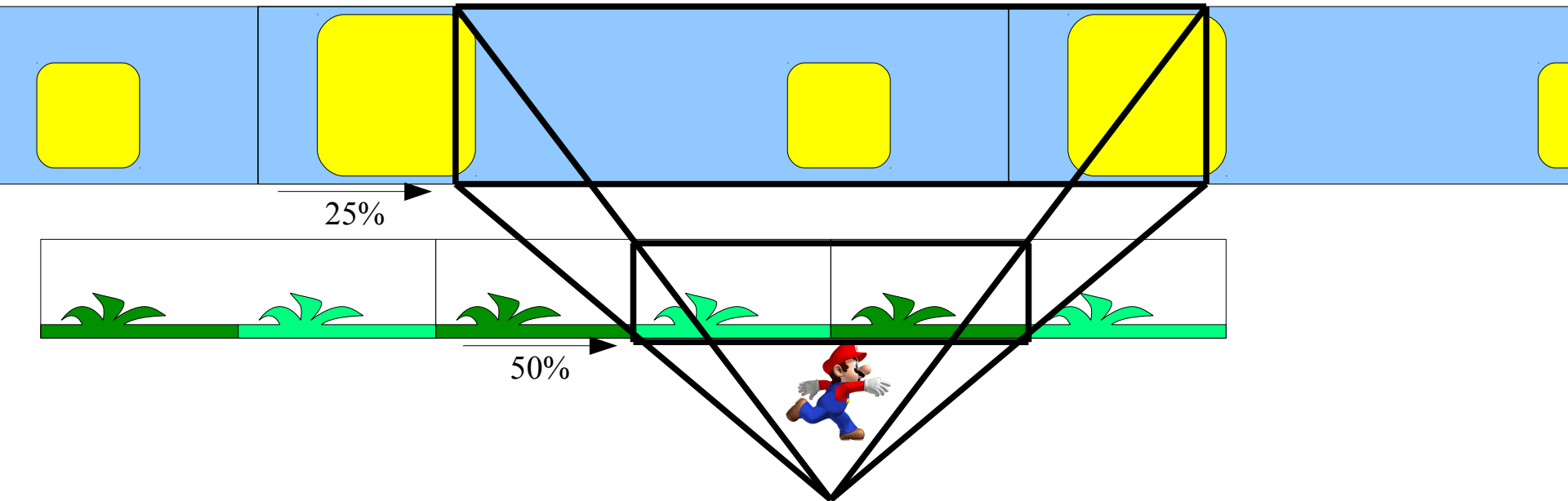
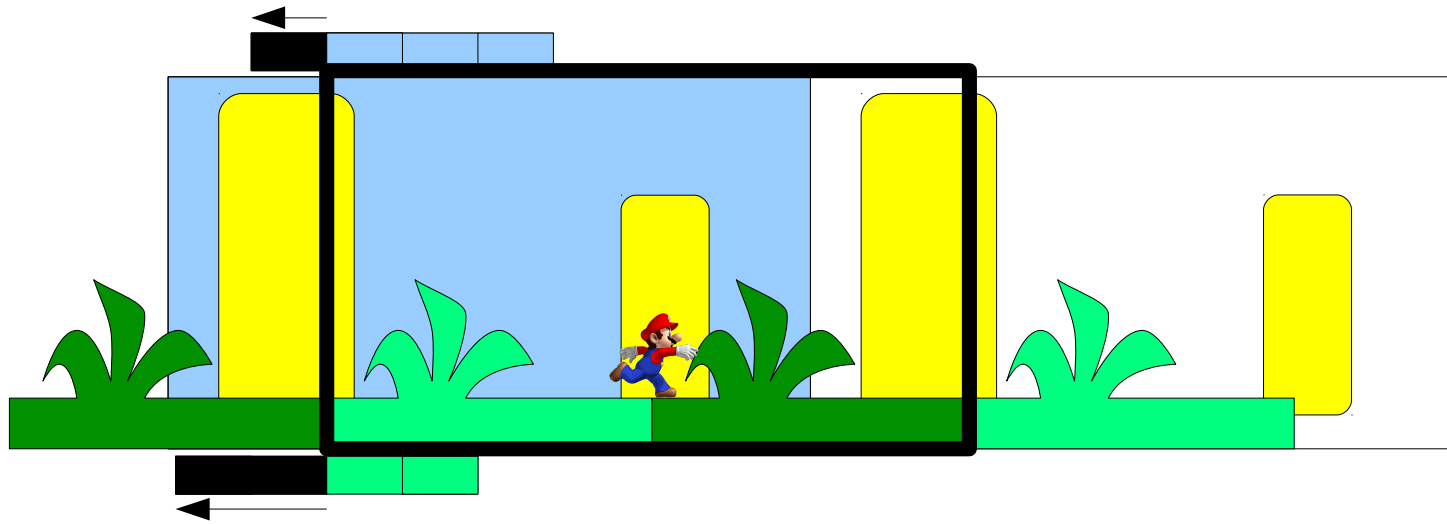
Défilement parallaxe



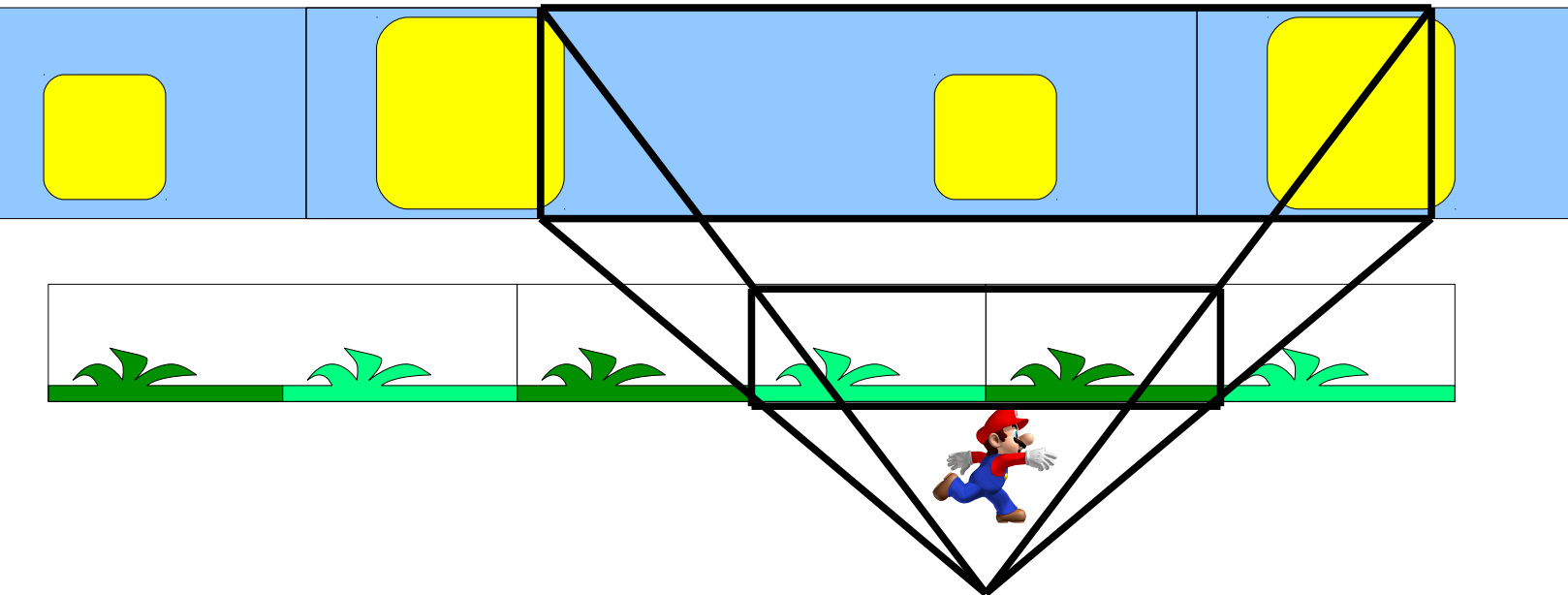
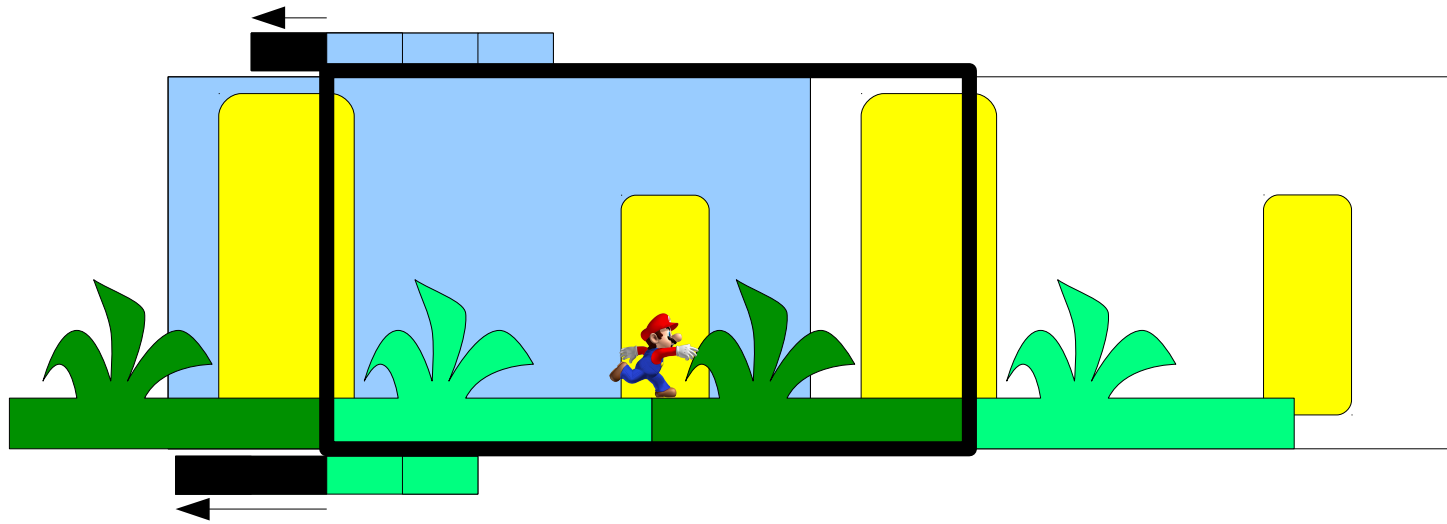
Défilement parallaxe



Défilement parallaxe



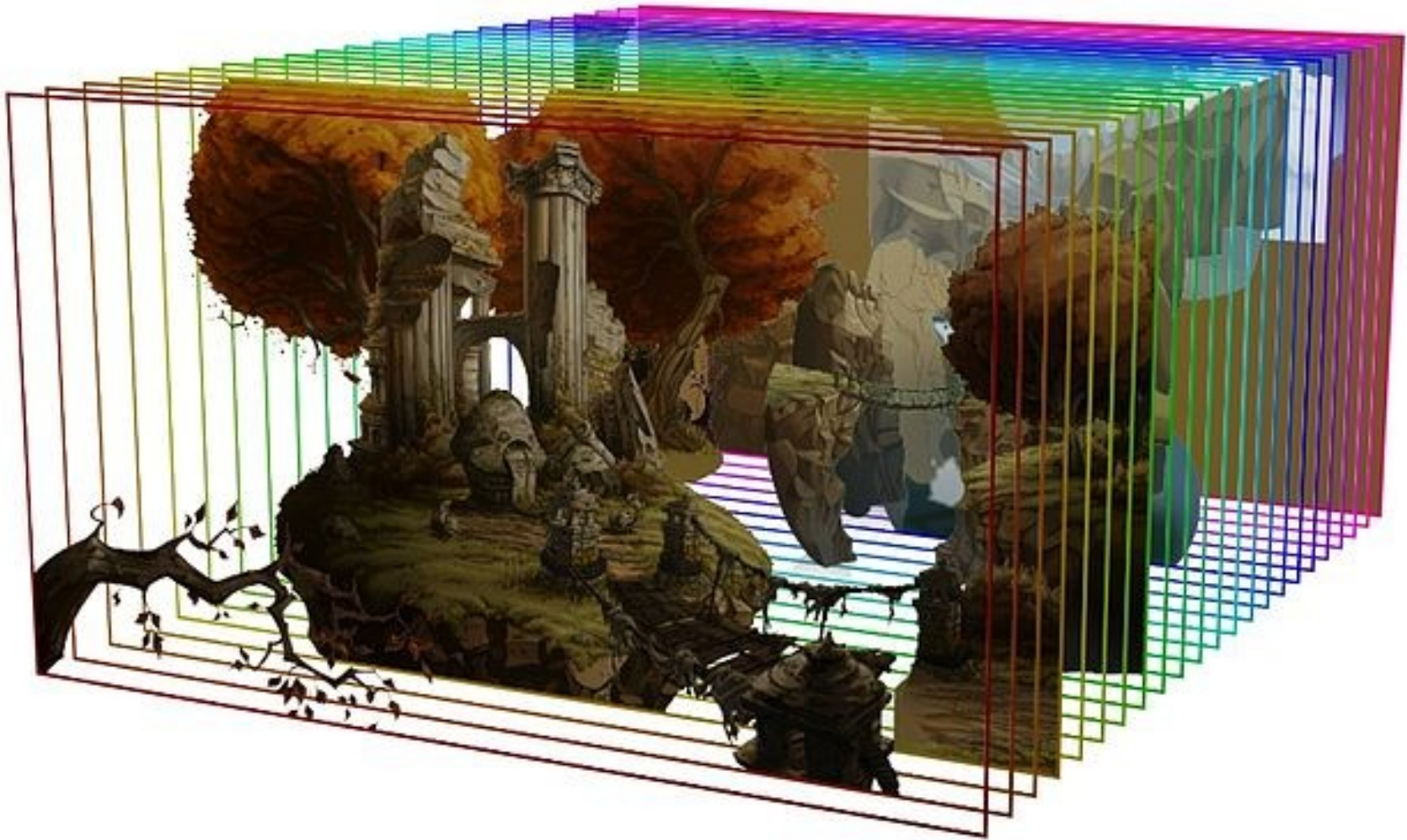
Défilement parallaxe



Vitesse $v/2$

Vitesse v

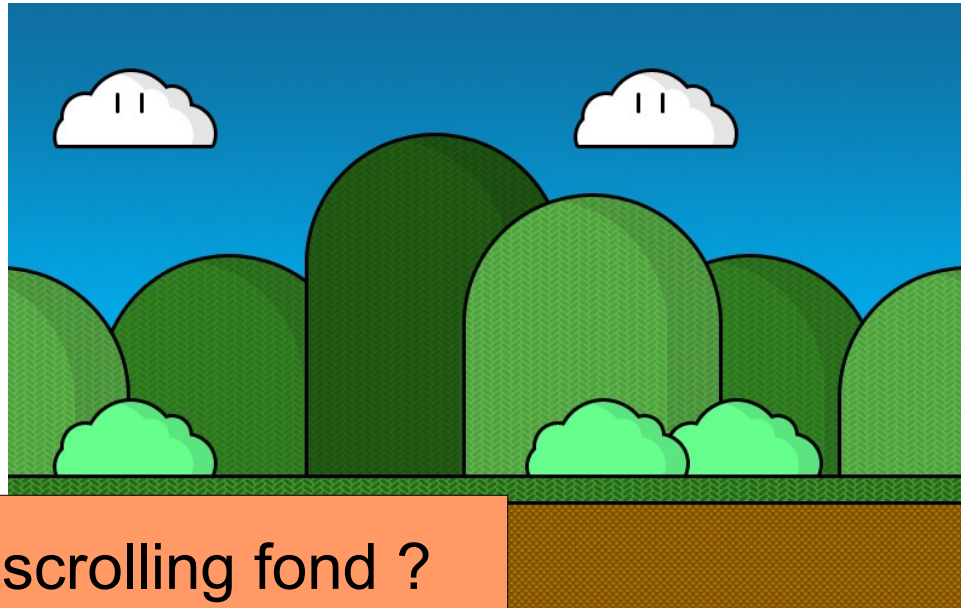
Parallaxe



<http://ollytyler.blogspot.fr>

Code Scrolling Parralaxe

- Scrolling



Comment gérer scrolling fond ?

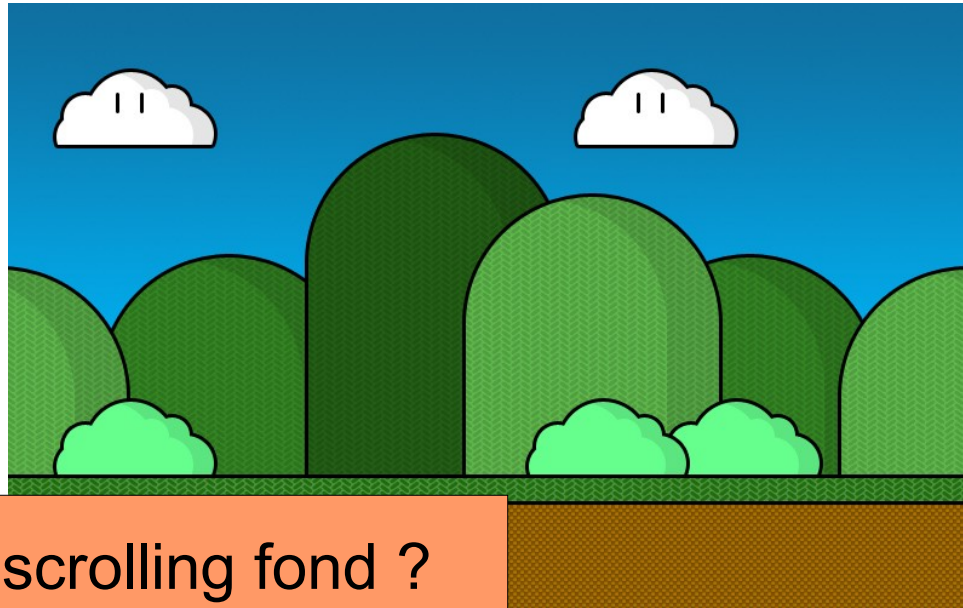
```
public void affiche(int x,Graphics g)
{
    //on se ramene au repère du plan
    x=(x%wx);

    //on affiche sur l'ecran 0 ==> wx-x image source de x à wx
    g.drawImage(im, 0 ,0 , wx-x, wy, x, 0, wx, wy,null);

    //on affiche sur l'ecran wx-x ==> wx image source de 0 à x
    g.drawImage(im, wx-x ,0 , wx, wy, 0, 0, x, wy,null);
}
```


Code Scrolling Parralaxe

- Scrolling



Comment gérer scrolling fond ?

x devient x/2



```
public void affiche(int x,Graphics g)
```

```
{
//on se ramene au repère du plan
x=(x/2 % wx);
```

```
    //pour l'ecran 0 ==> wx-x image source de x à wx
    g.drawImage(im, 0, 0, wx-x, wy, x, 0, wx, wy,null);
```

```
    //pour l'ecran wx-x ==> wx image source de 0 à x
    g.drawImage(im, wx-x, 0, wx, wy, 0, 0, x, wy,null);
```

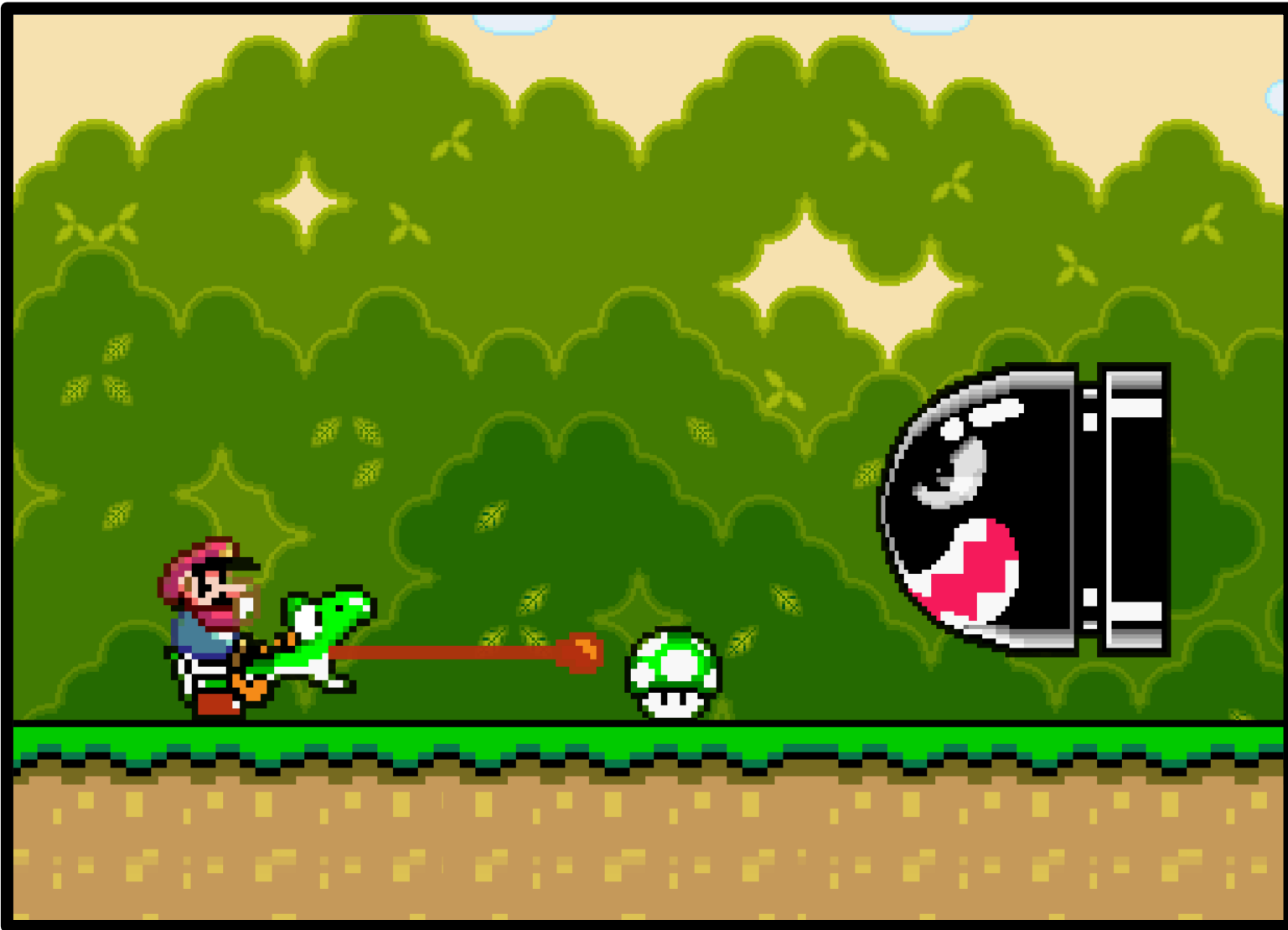
```
}
```

Meme modulo = meme taille
x=x/2 ⇒ déplacement moins rapide

Démonstration

Défilement Parallaxe

- **Modifications des repères de jeu**
 - Suivi vertical
 - Zoom
 - Changement vitesse
- **Démonstrations**

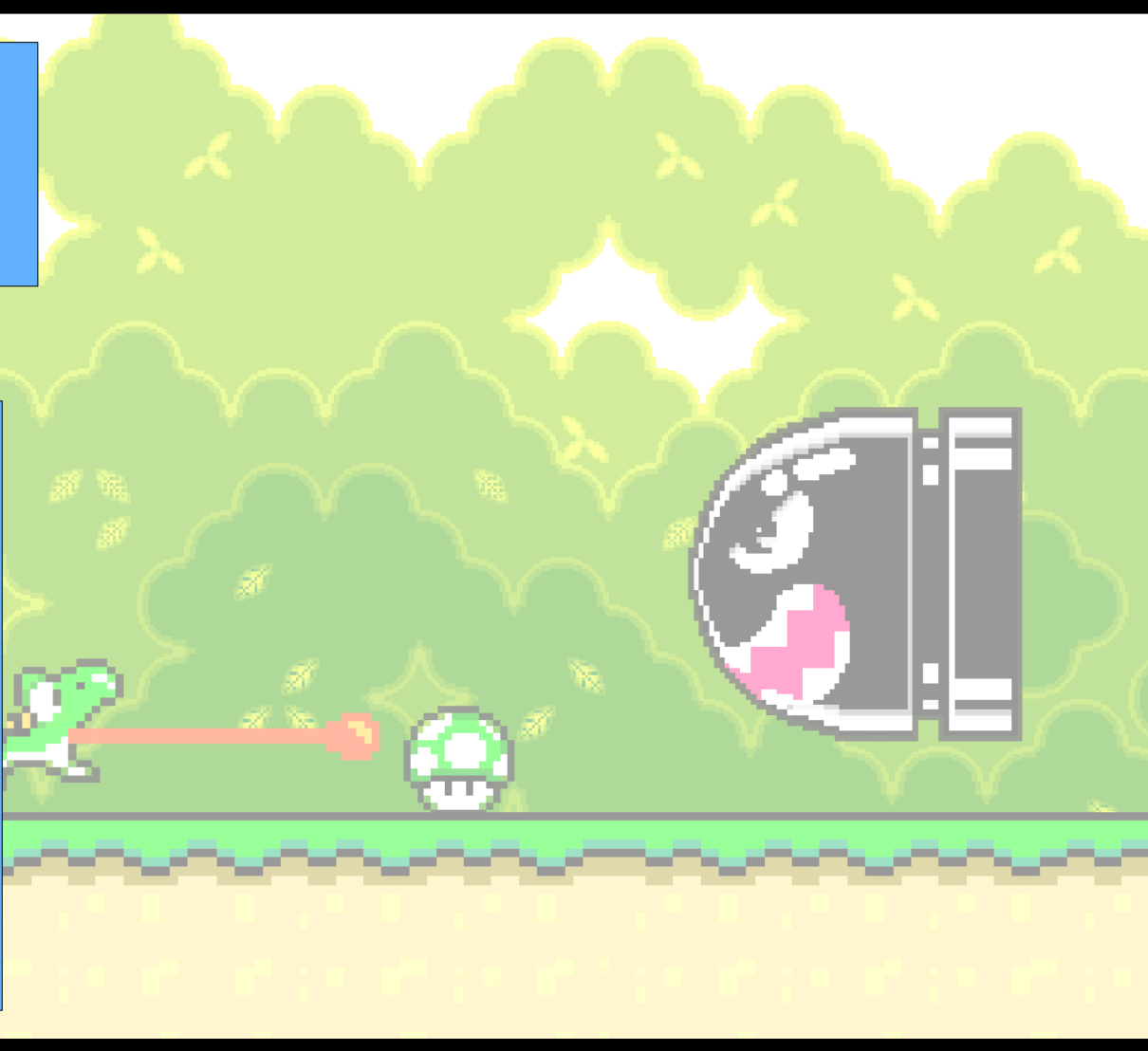


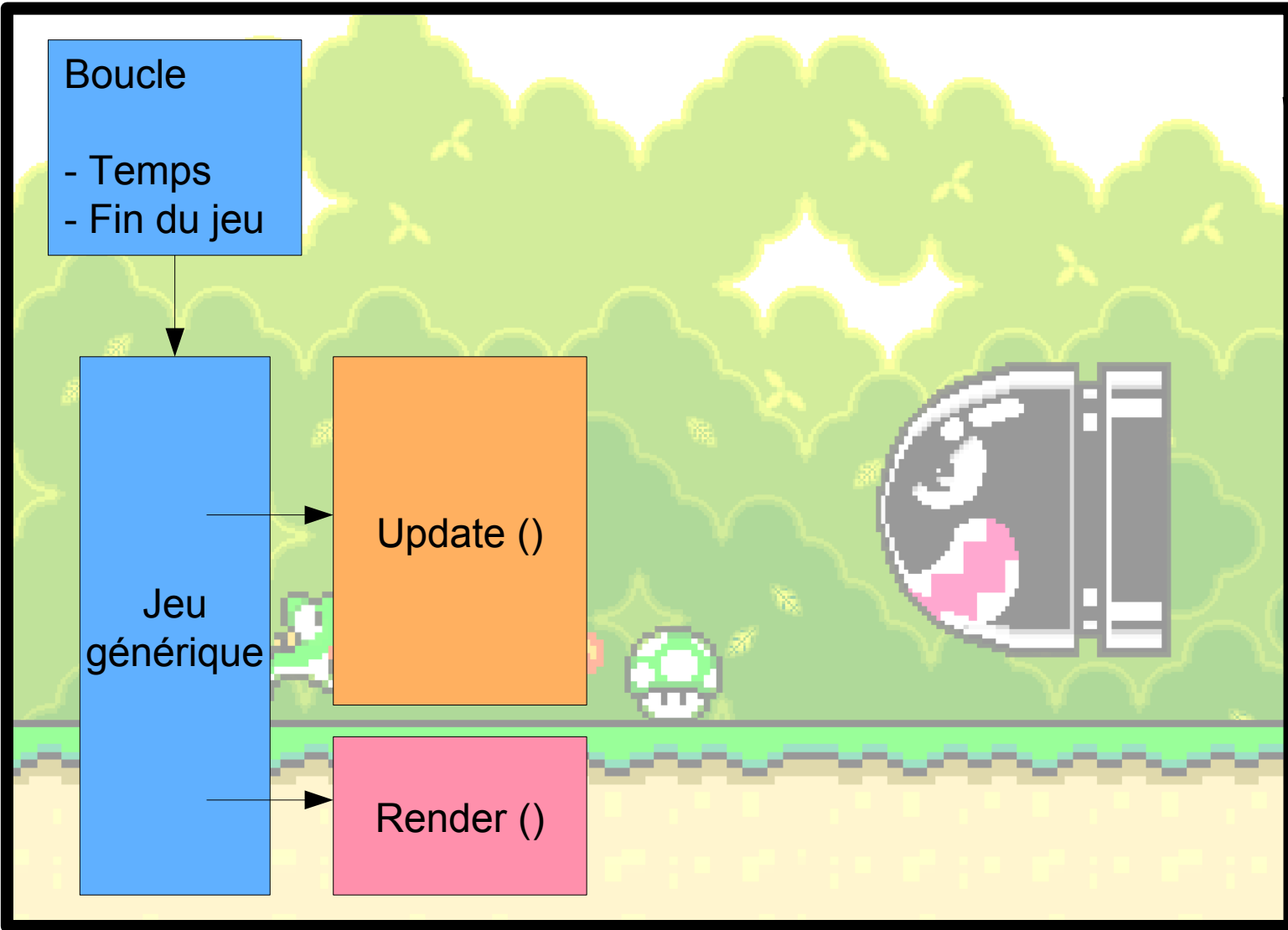


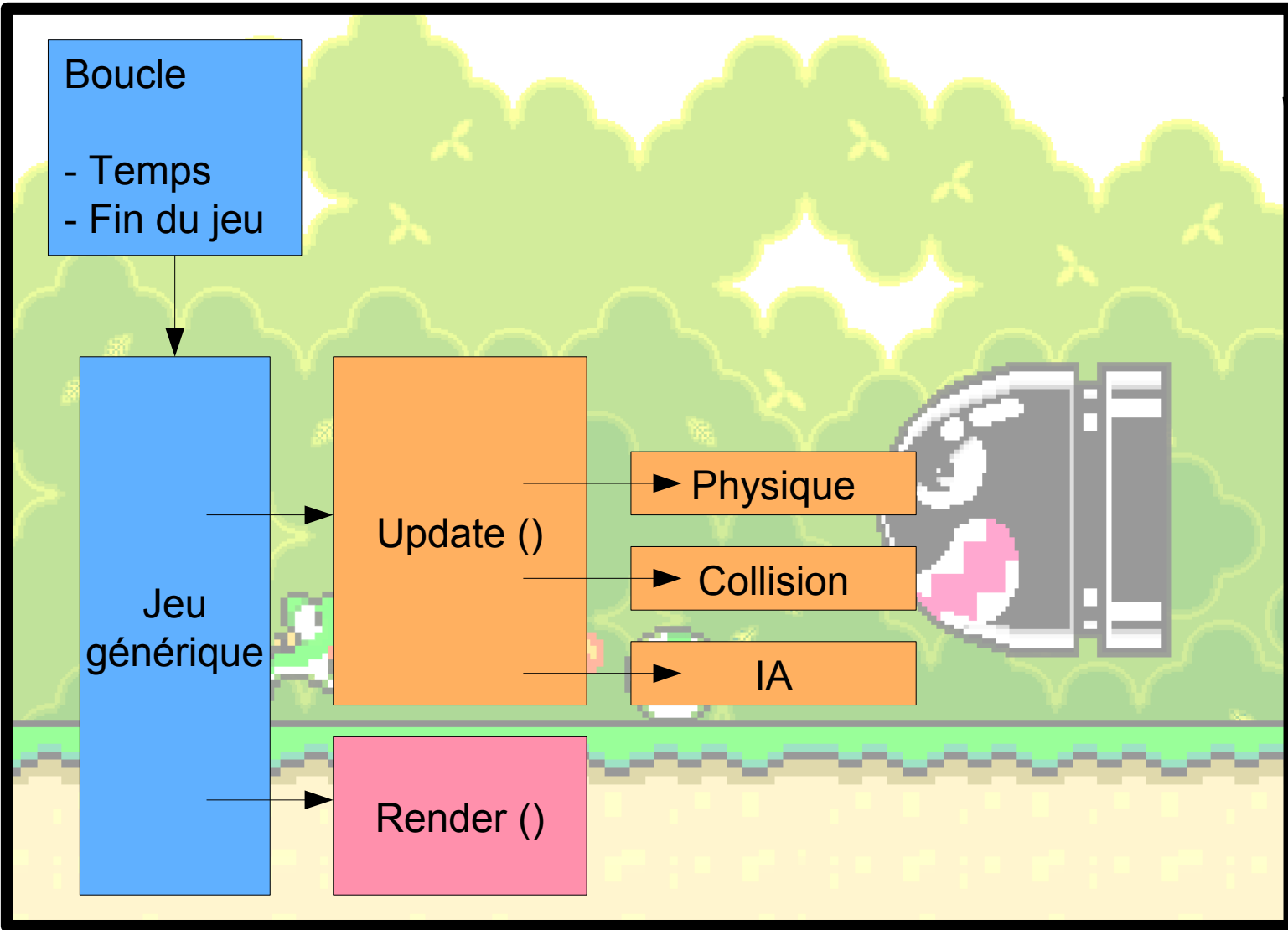
Boucle

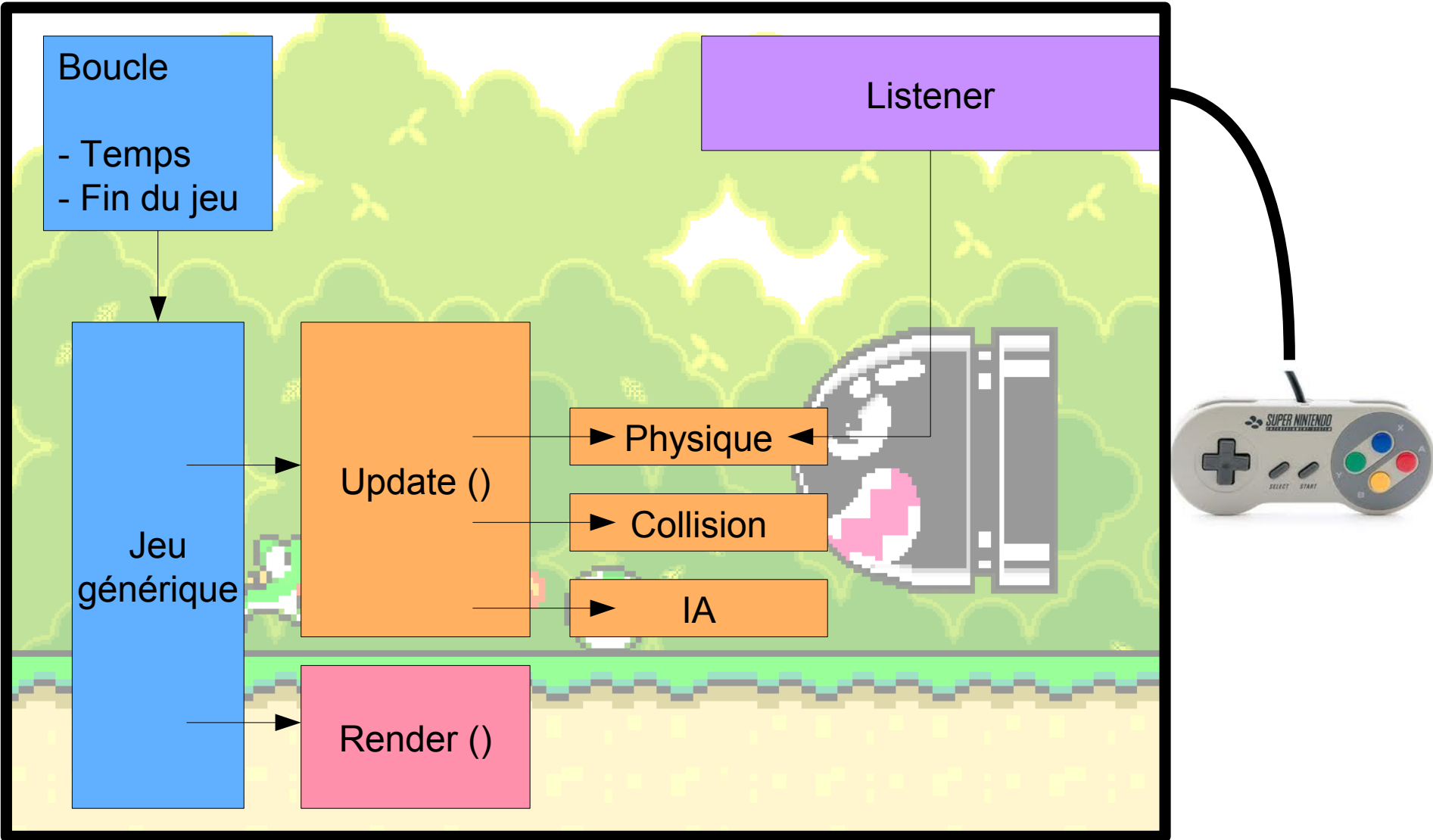
- Temps
- Fin du jeu

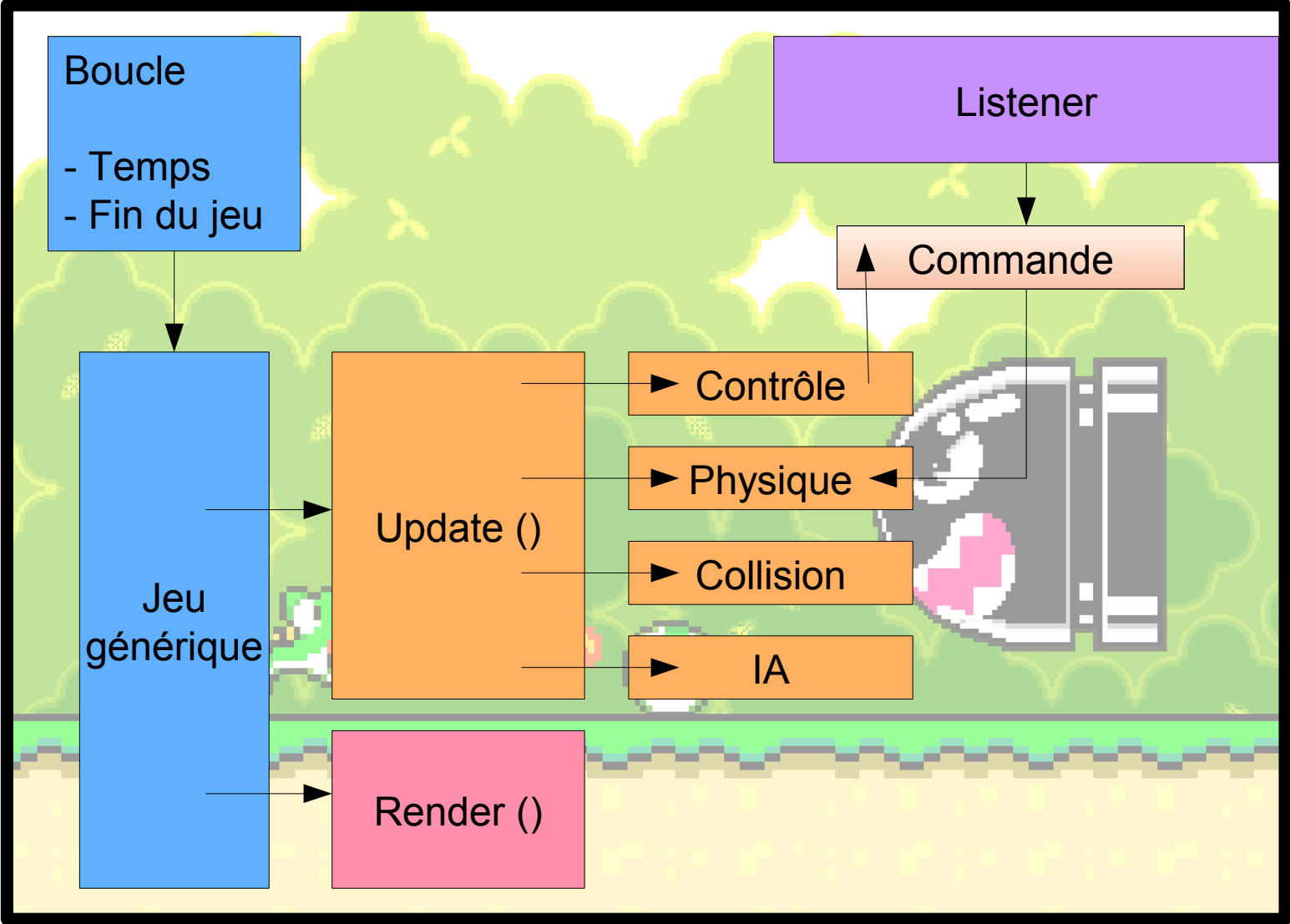
Jeu
générique

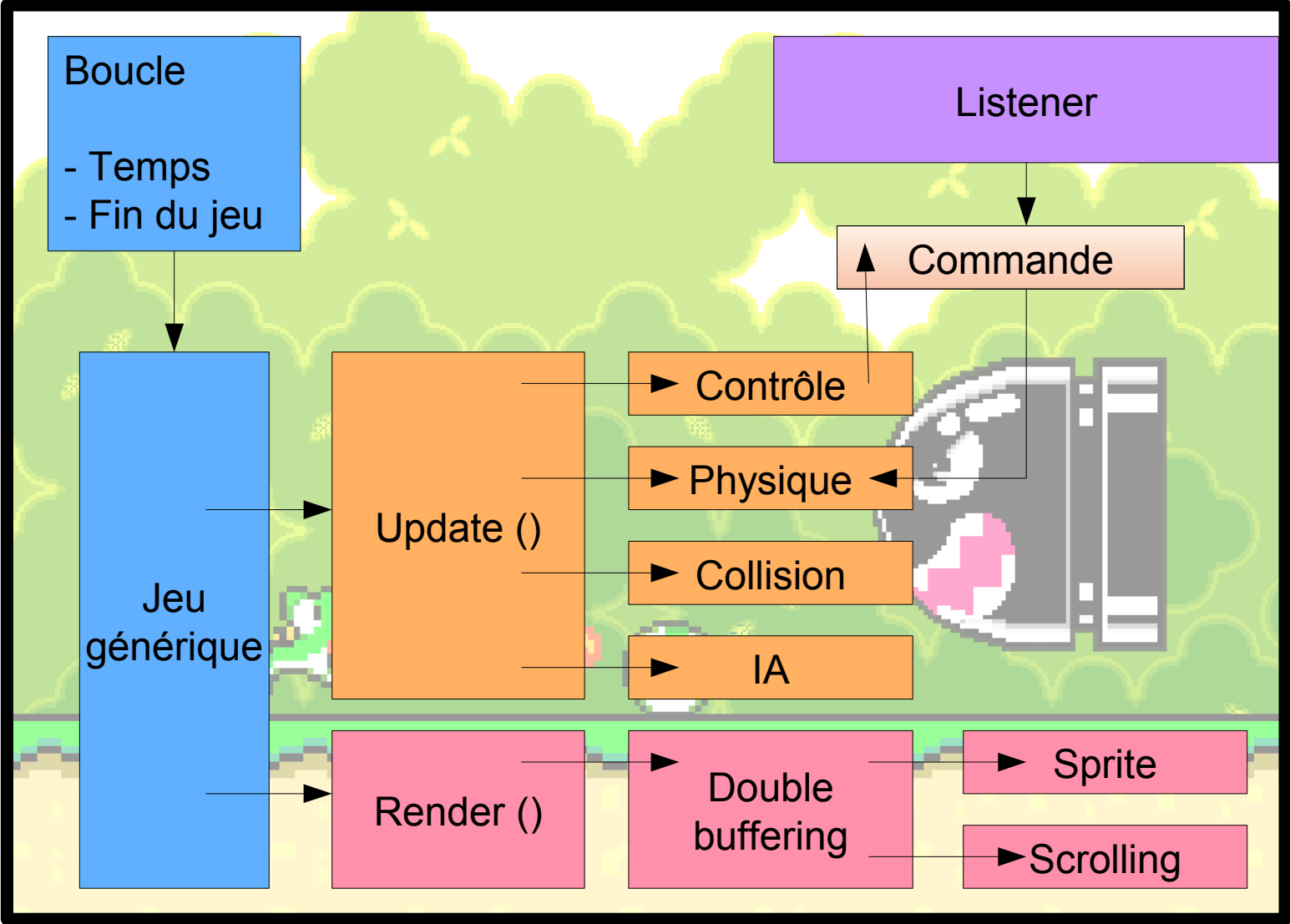












- **Bien penser l'organisation de son code**
 - Qui a accès à quoi ?
 - Exemple activités (render/update), controleur,...
- **Avoir une vision d'ensemble**
 - Après avoir une idée de prototype
- **Tester progressivement**
 - Avancer de manière incrémentale (agile)

- **Internet**

- Killer game programming in java

- <http://fivedots.coe.psu.ac.th/~ad/jg/> (chap 1 et 2)

- **Livre**

- Killer Game Programming in Java

- by Andrew Davison

- Developing Games In Java

- By David Brackeen, Bret Barker, Laurence Vanhelsuwé