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

Février 2013

Introduction

- En 1990 (il y a 23 ans)

Introduction

- En 1990

1987

Introduction

- En 1990

1987

Introduction

- En 1990

Introduction

- En 1990

CPC 6128

Introduction

- En 1990

Introduction

Nouveau JEU révolutionnaire !!!

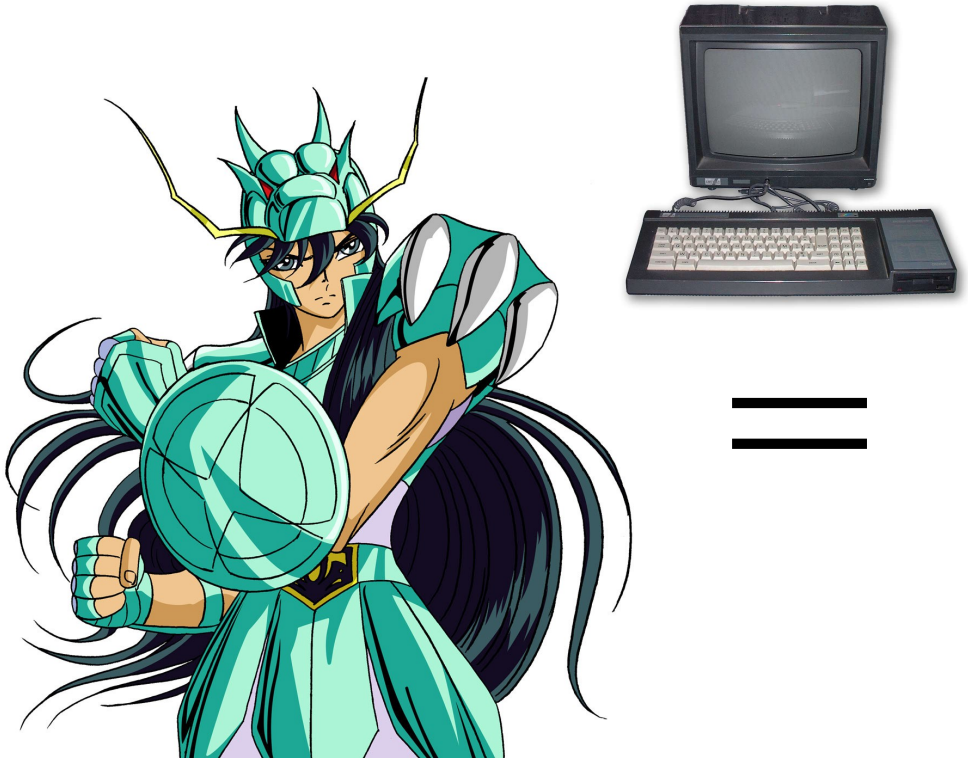
Introduction

- En 1990

Introduction

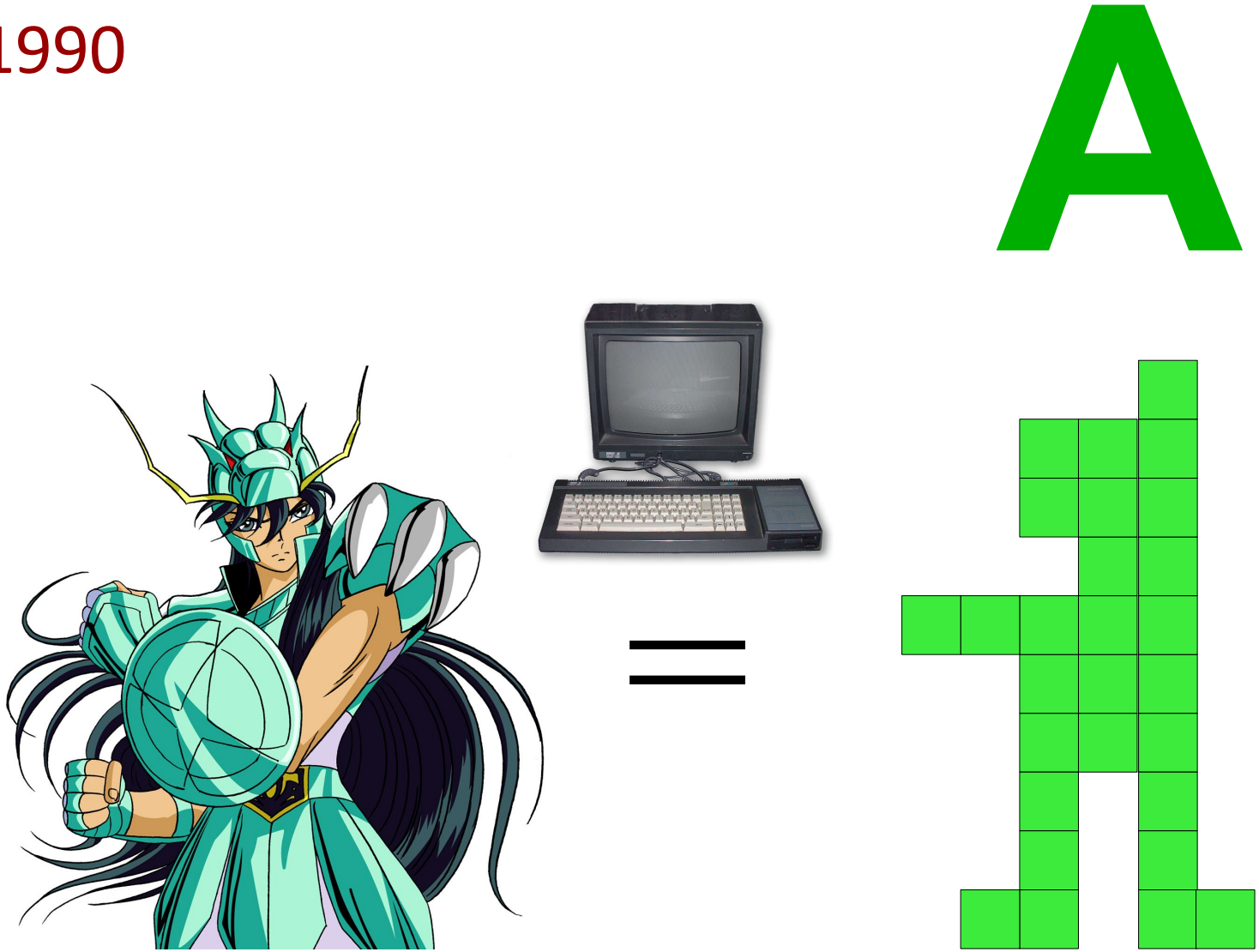
- En 1990

A



Introduction

- En 1990



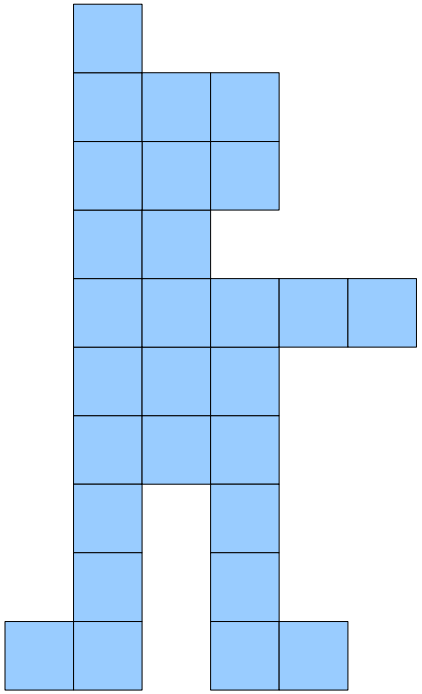
Introduction

- En 1990

=

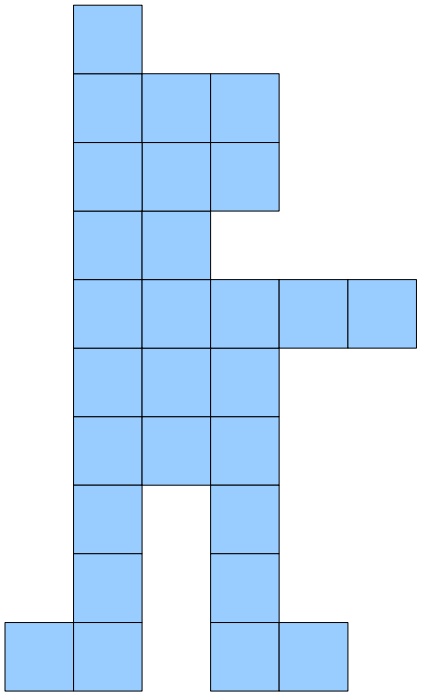
Introduction

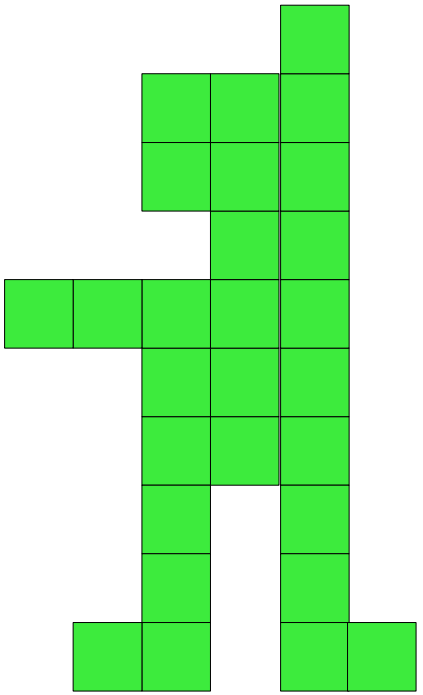
- En 1990



Introduction

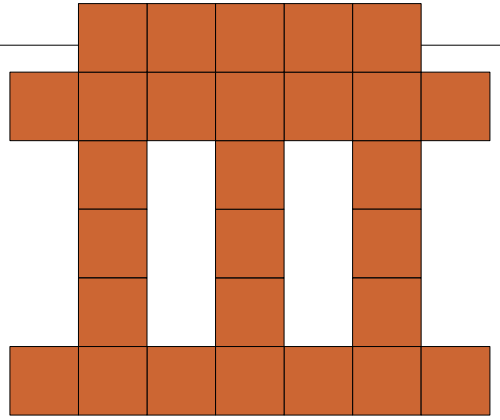
- En 1990

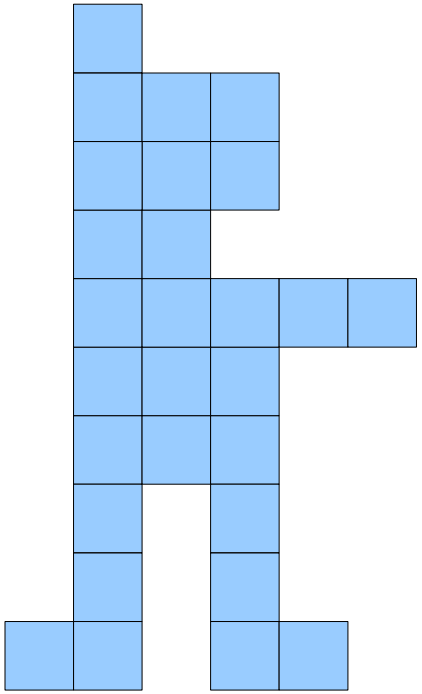


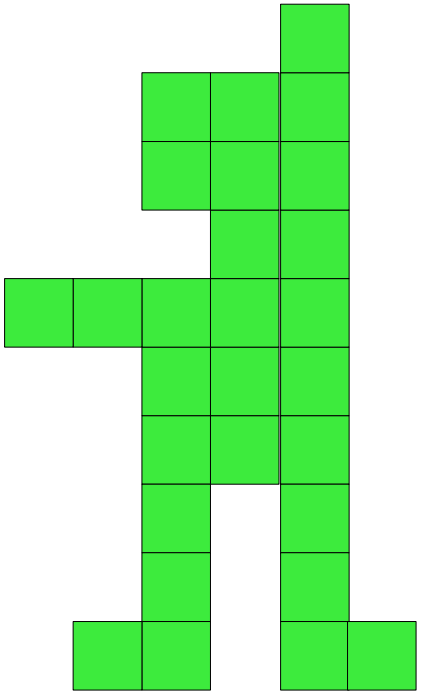


Introduction

- En 1990



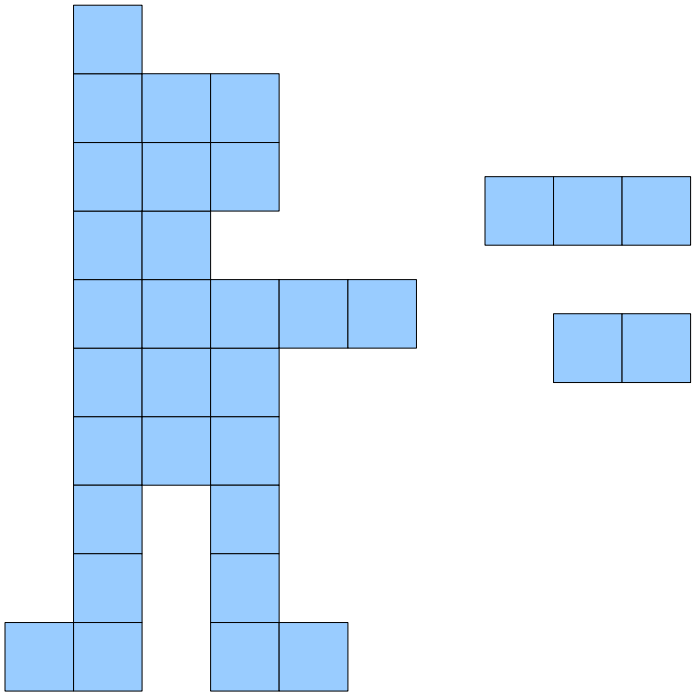


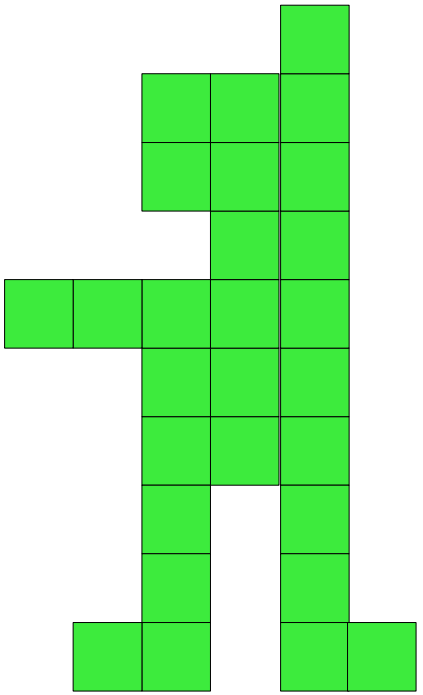


Introduction

- En 1990

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

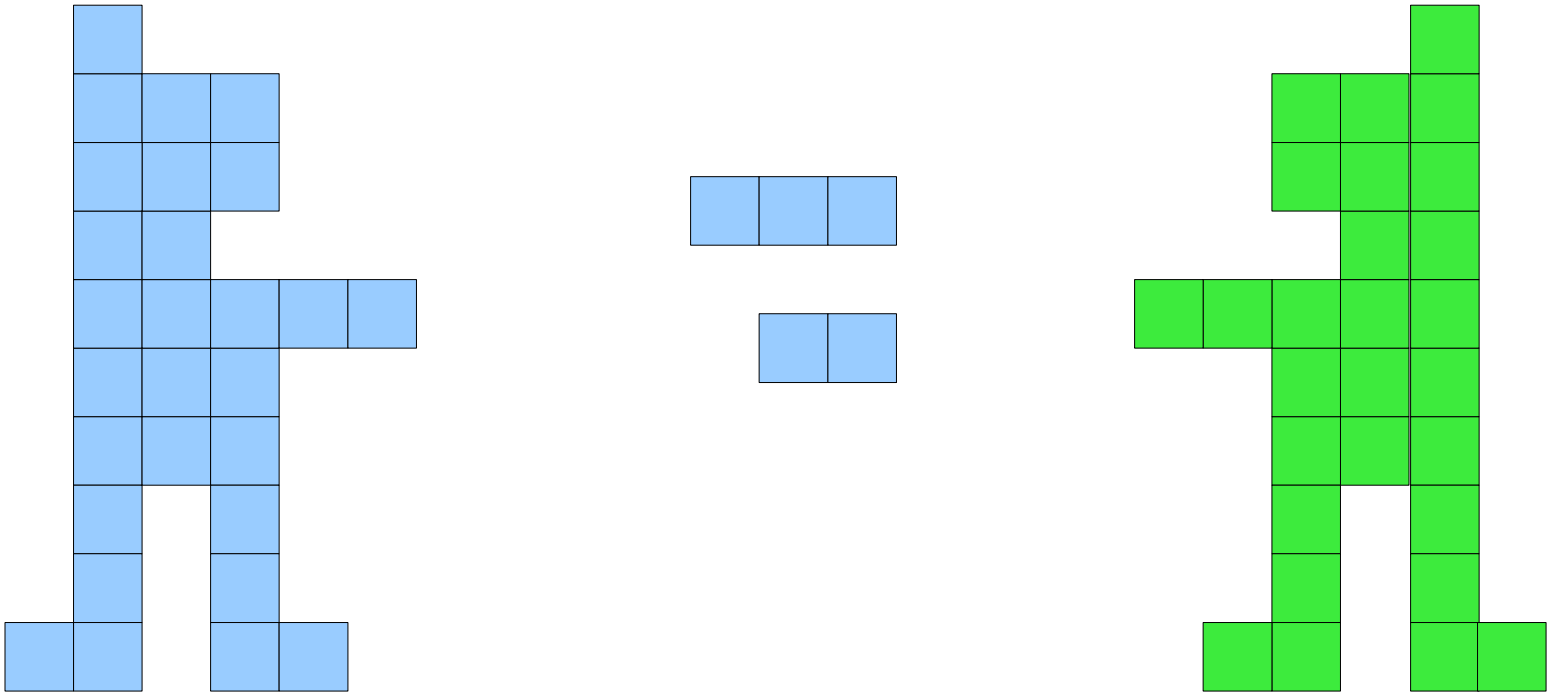




Introduction

- En 1990

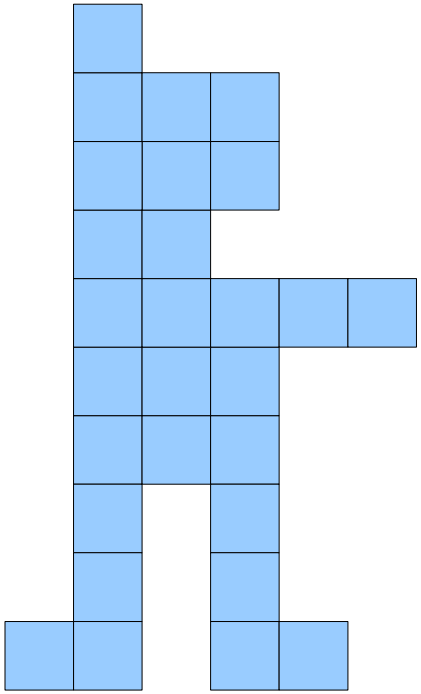
```
10 : i = 0
20 : i++
30 : If (i<40) GOTO 10
```

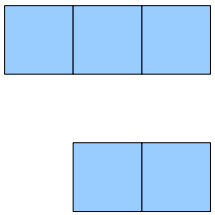


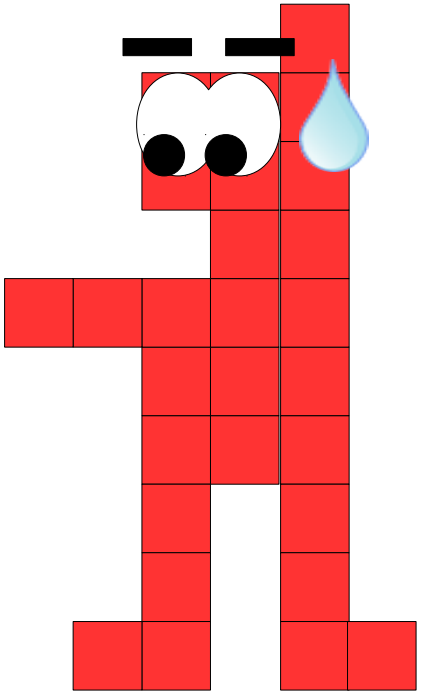
Introduction

- En 1990

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



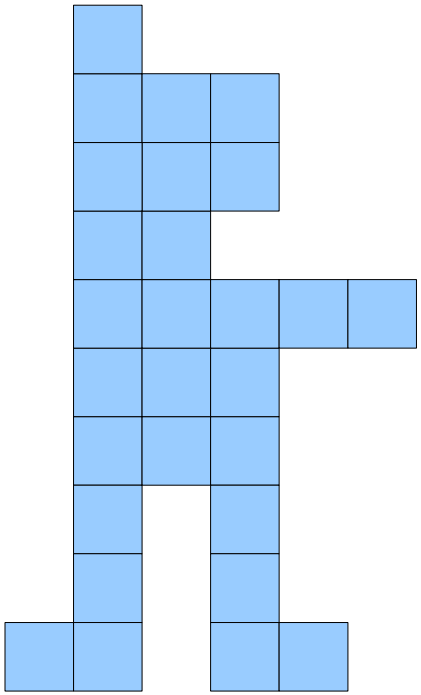


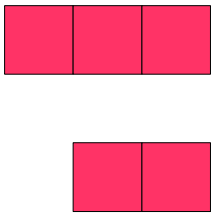


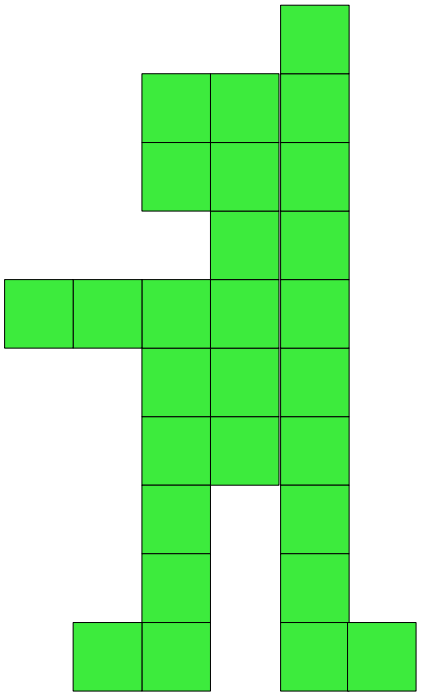
Introduction

- En 1990

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



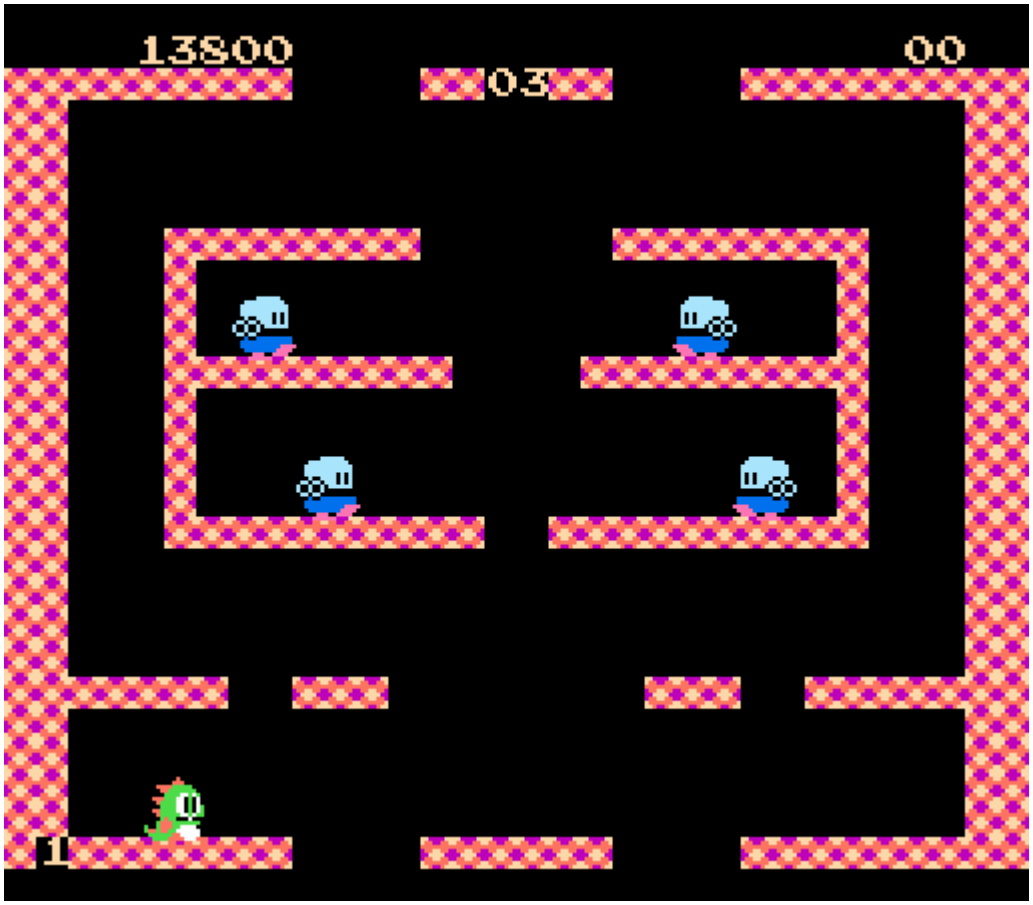




Introduction

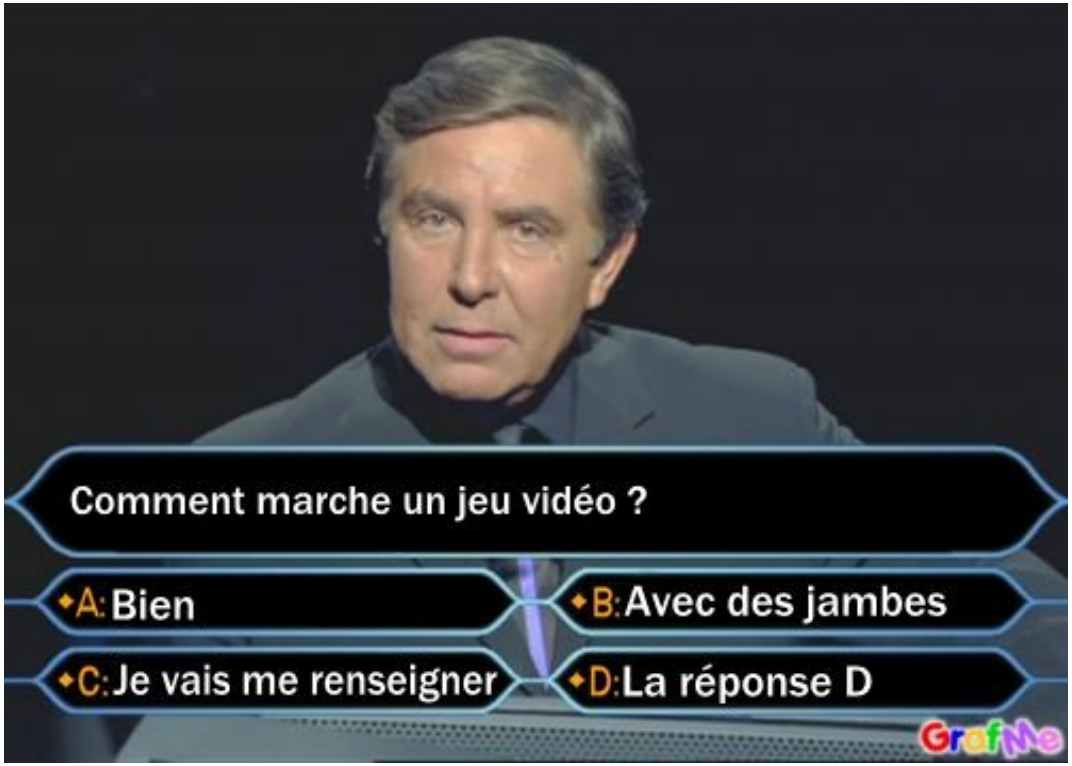
- En 1990

??????????



Introduction

- En 1990
- En 2005, ...



Introduction

- En 1990
- En 2005, ...

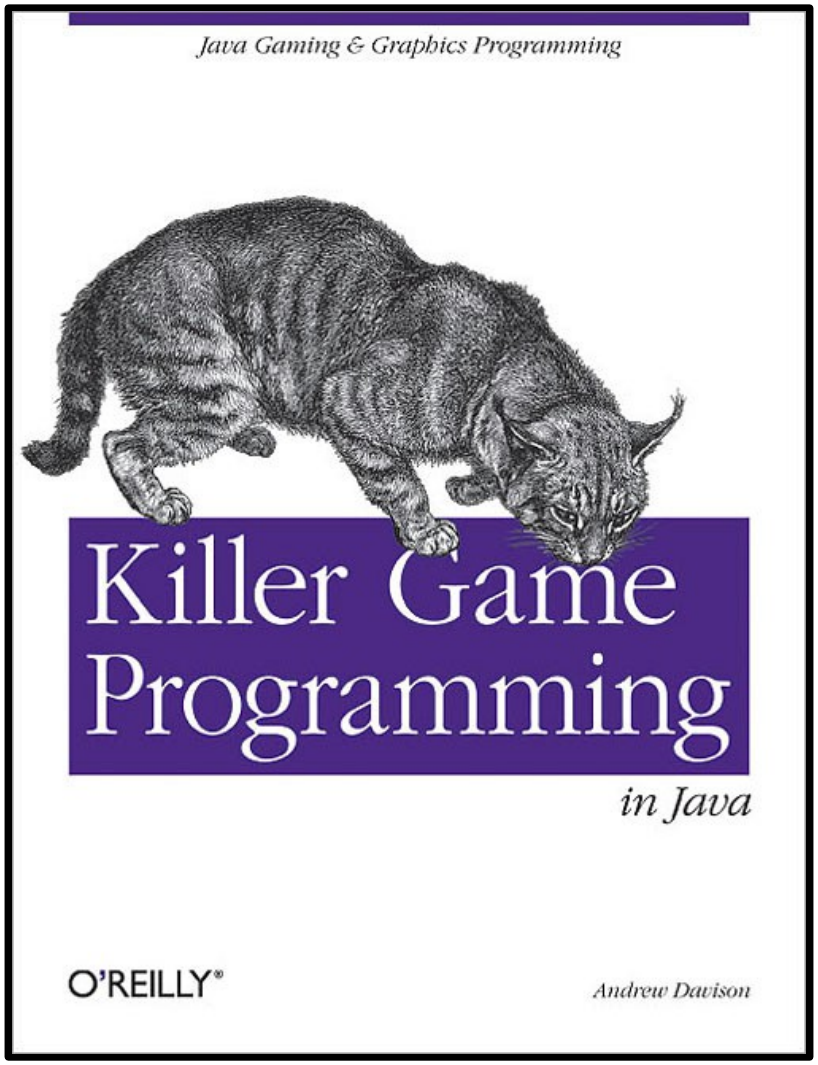
Introduction

- En 1990
- En 2005, ...

Introduction

- En 1990
- En 2005, ...

Une (bonne) référence



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

Plan

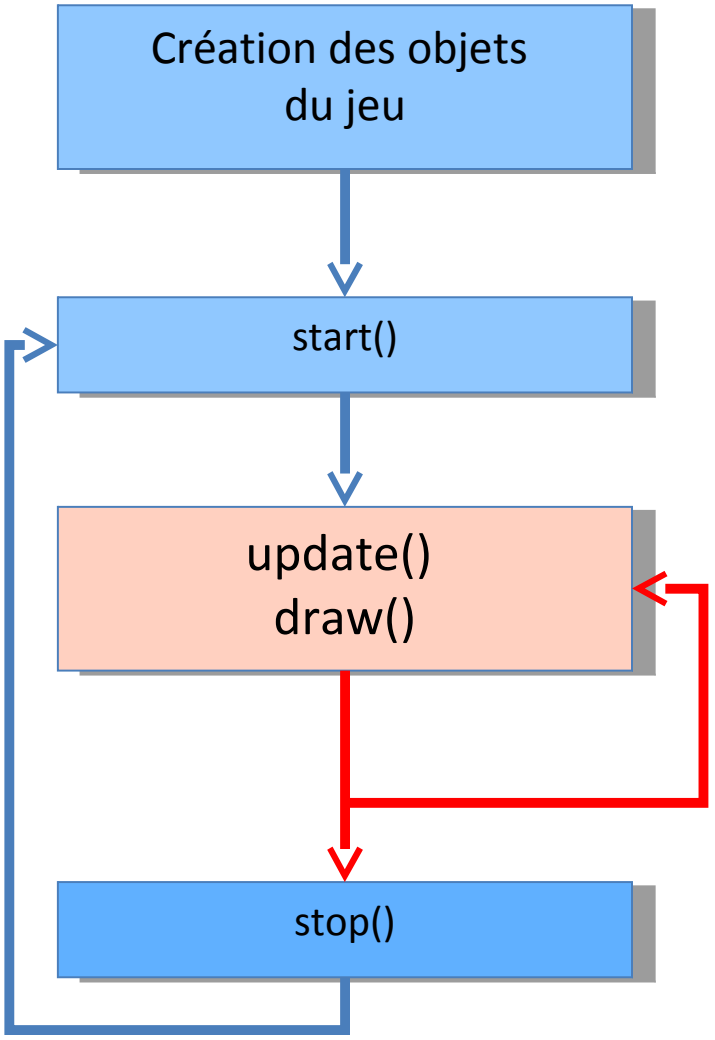
- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Contrôleur
- Affichage
- Réseau

Application Fil Rouge

Plan

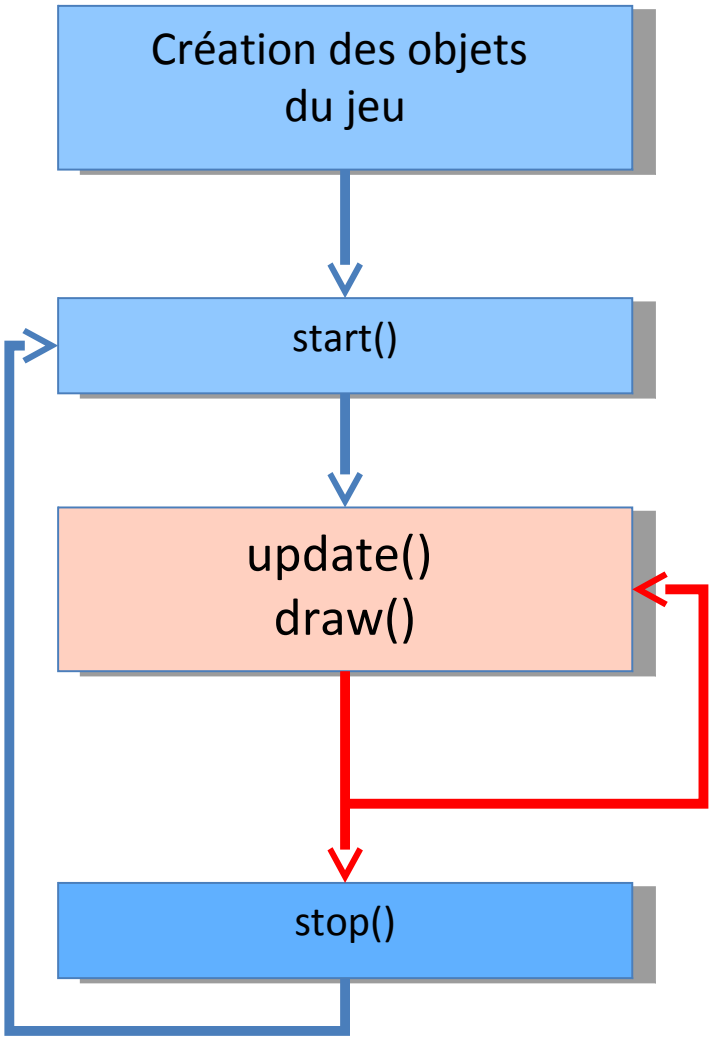
- Boucle de jeu
- Gestion du temps
- Modèle de jeu
- Gestion du Controleur
- Affichage
- Réseau

Boucle de jeu



Boucle principale du jeu

Boucle de jeu



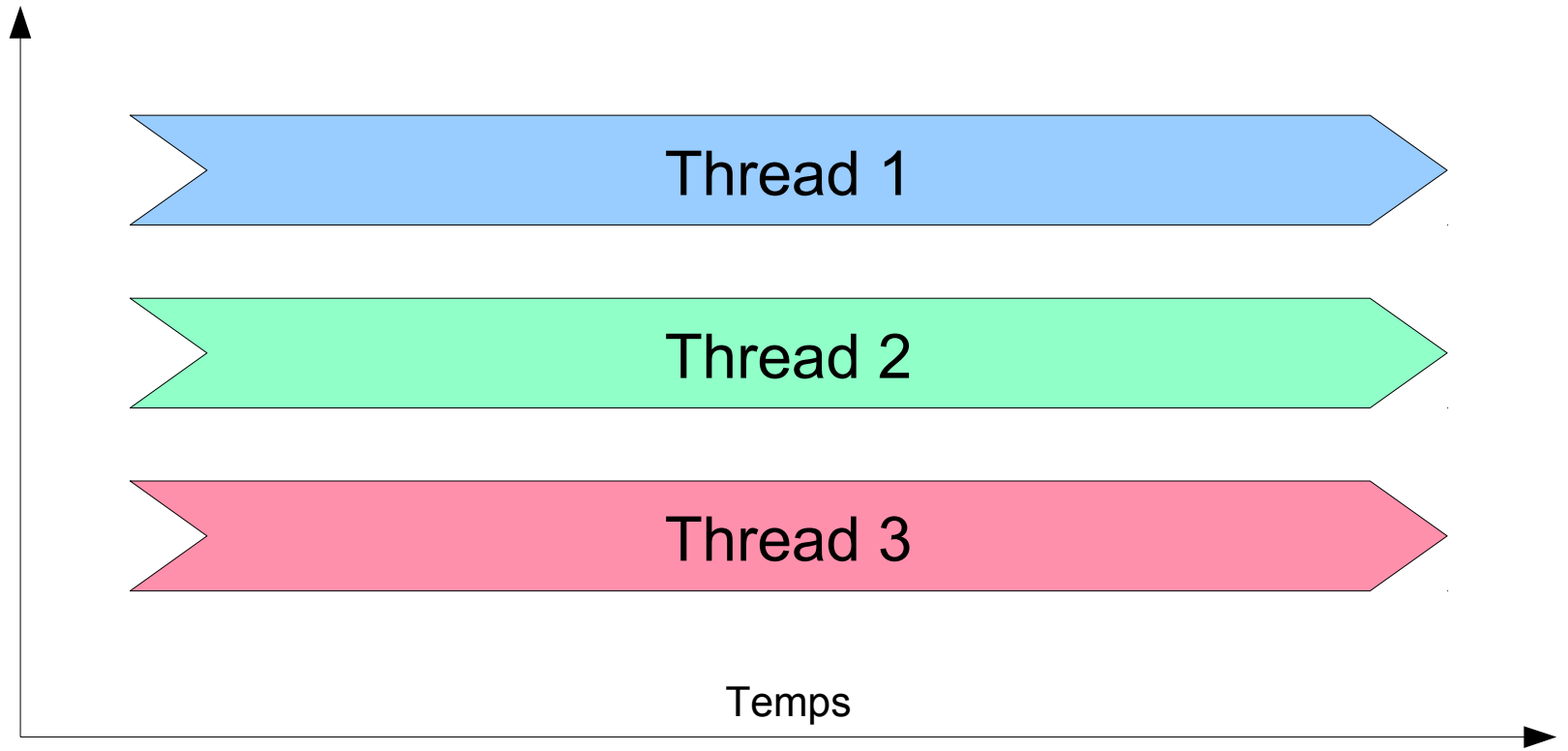
FPS

Frames par seconde

Objectif 60 fps

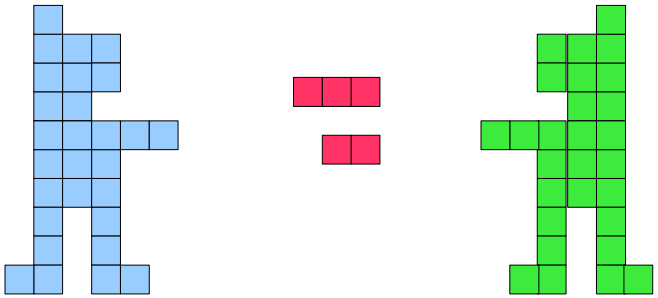
Thread

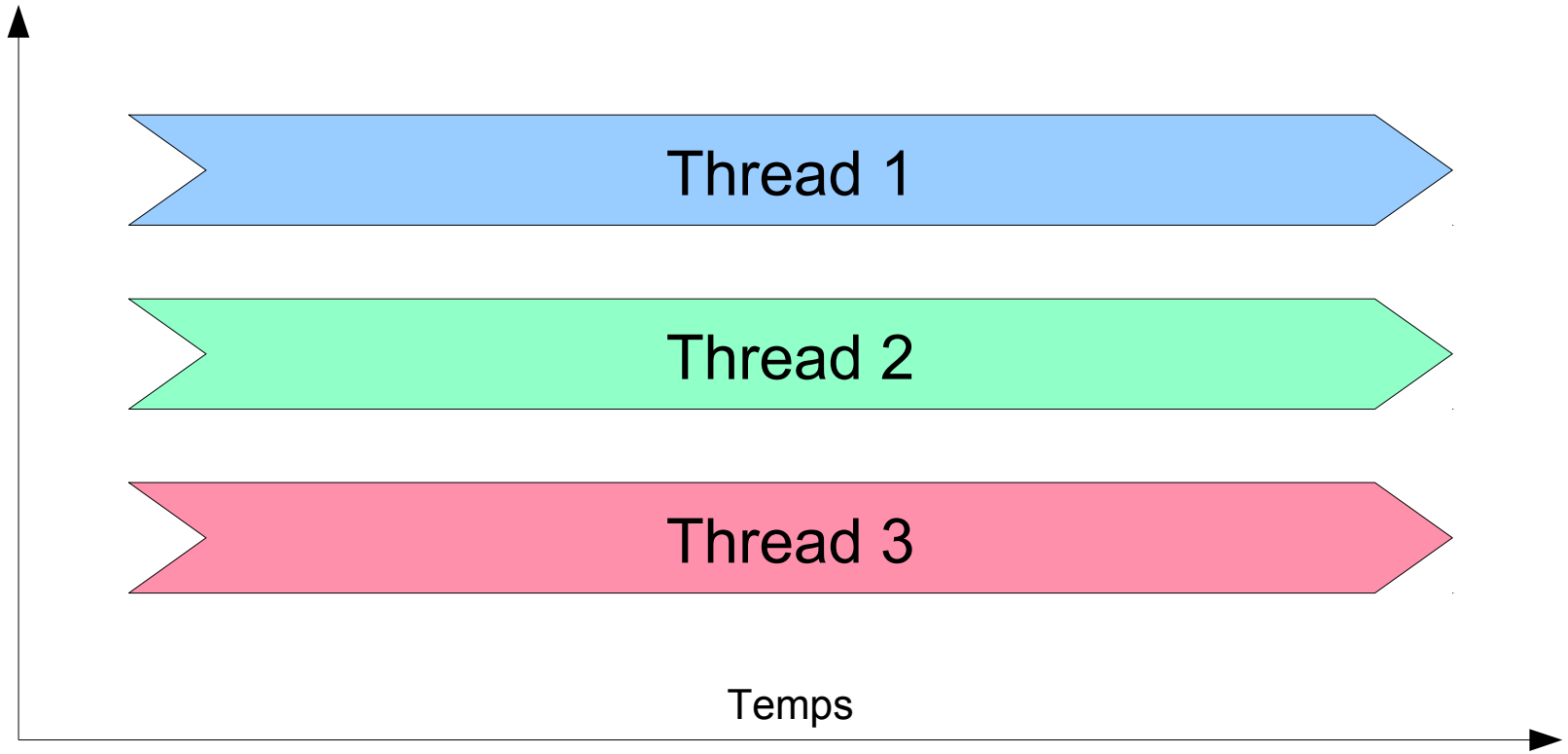
- Thread
  - Processus exécuté en parallèle
  - Des thread de base (ex JComponent)



Thread

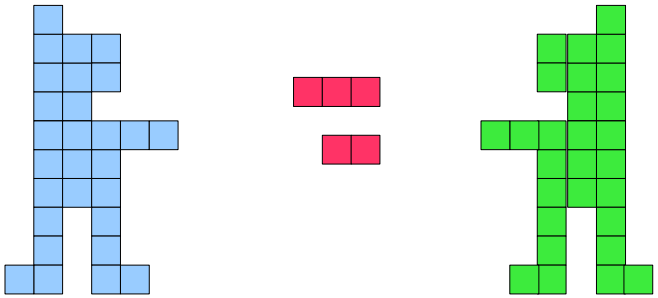

```
10 : i = 0
20 : i++
30 : If (i<40) GOTO 10
```

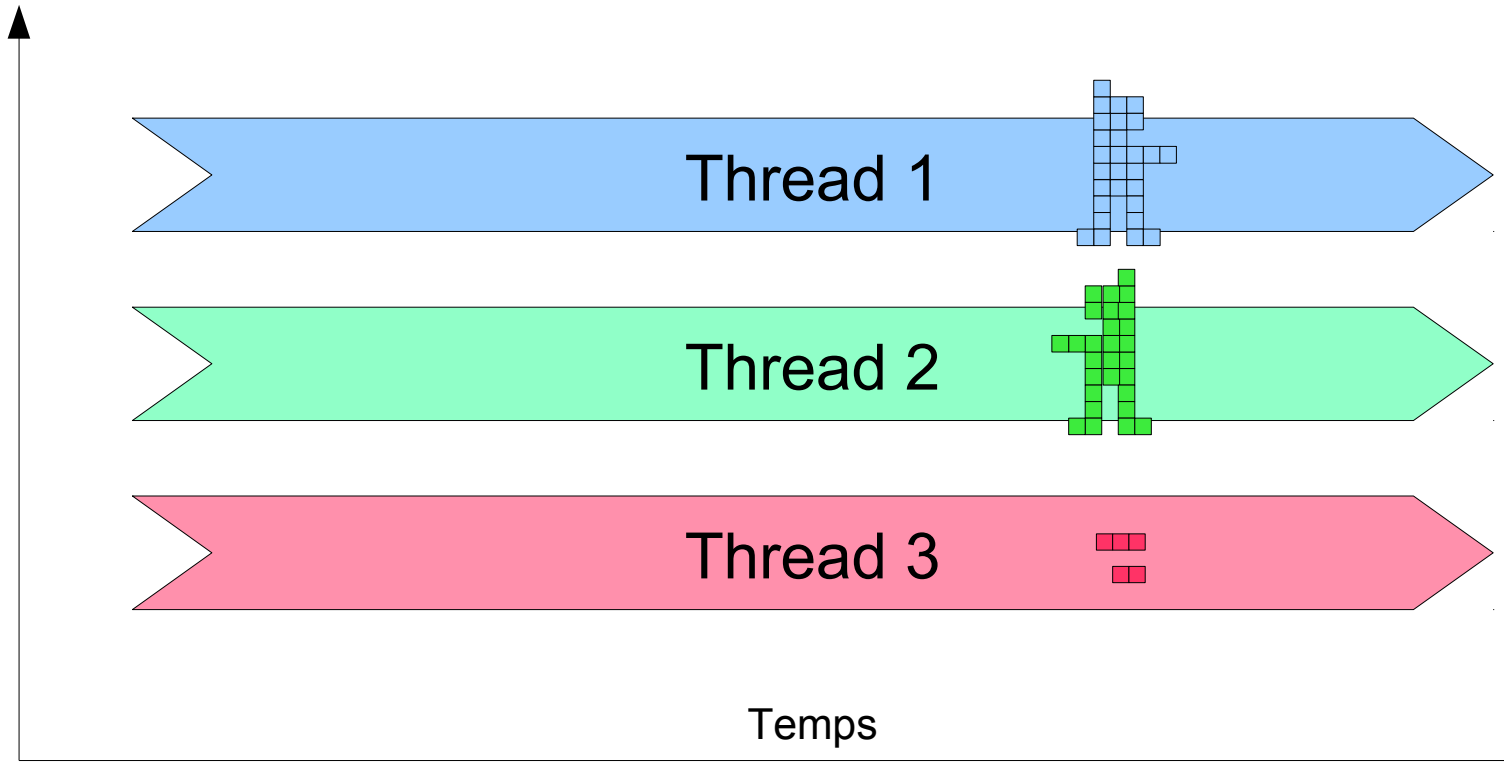




Thread


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

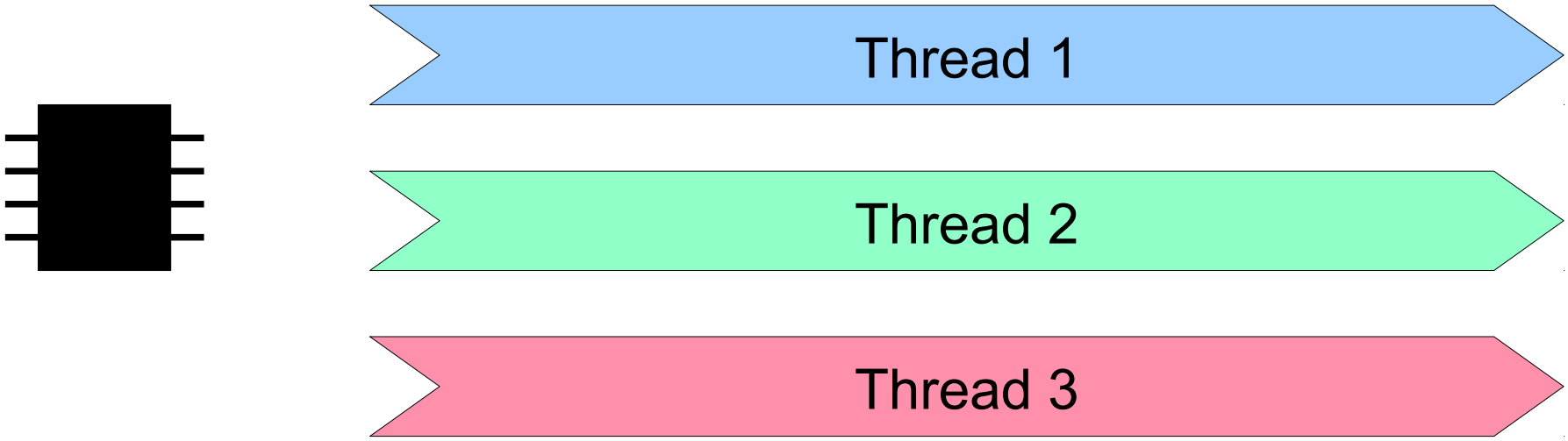




Thread

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

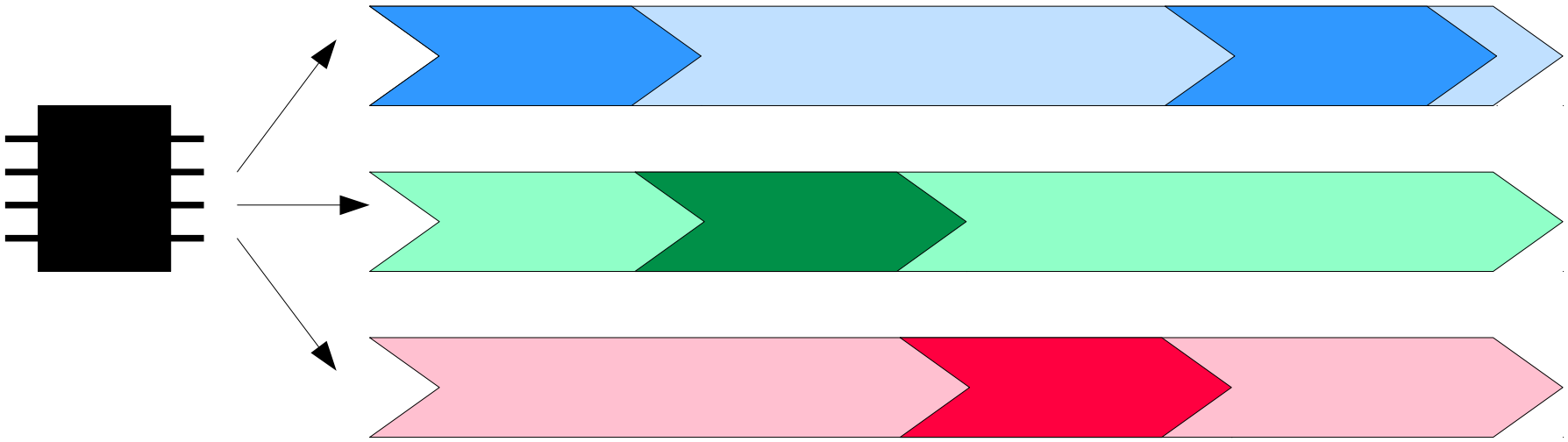
abstraction



Thread

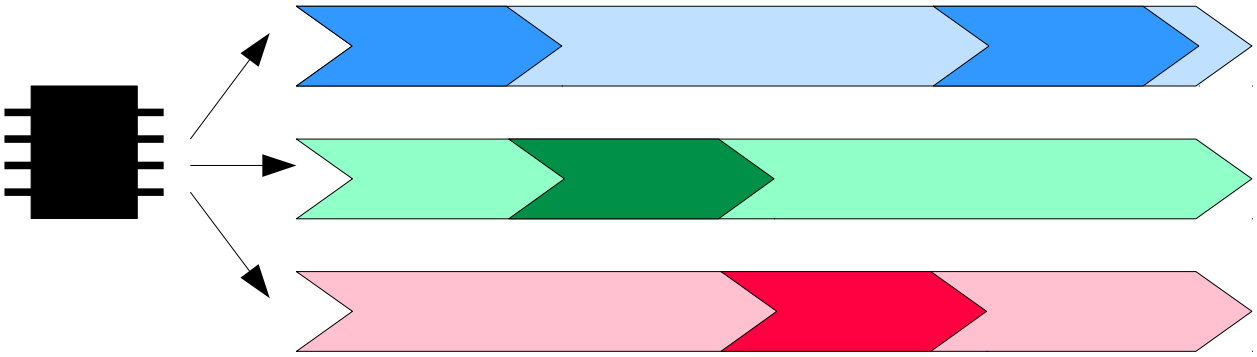
- Thread
  - Processus exécuté en parallèle
  - Des thread de base (ex JComponent)

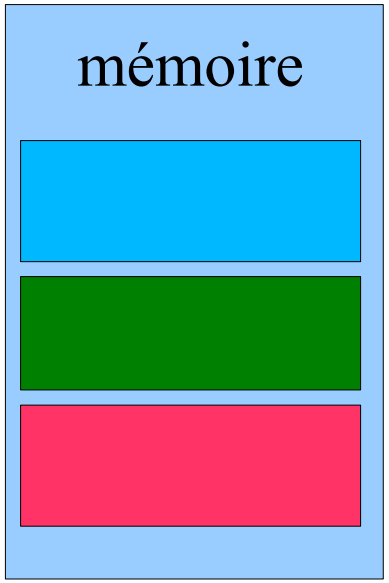
abstraction



Thread

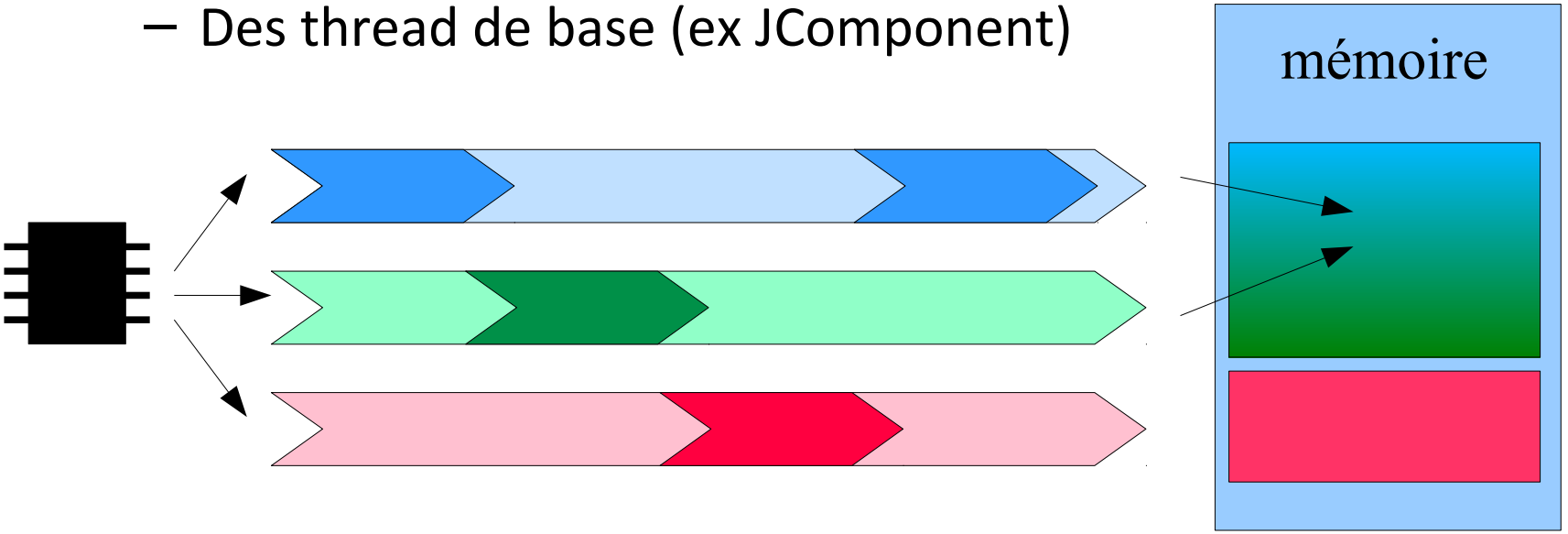
- Thread
  - Processus exécuté en parallèle
  - Des thread de base (ex JComponent)





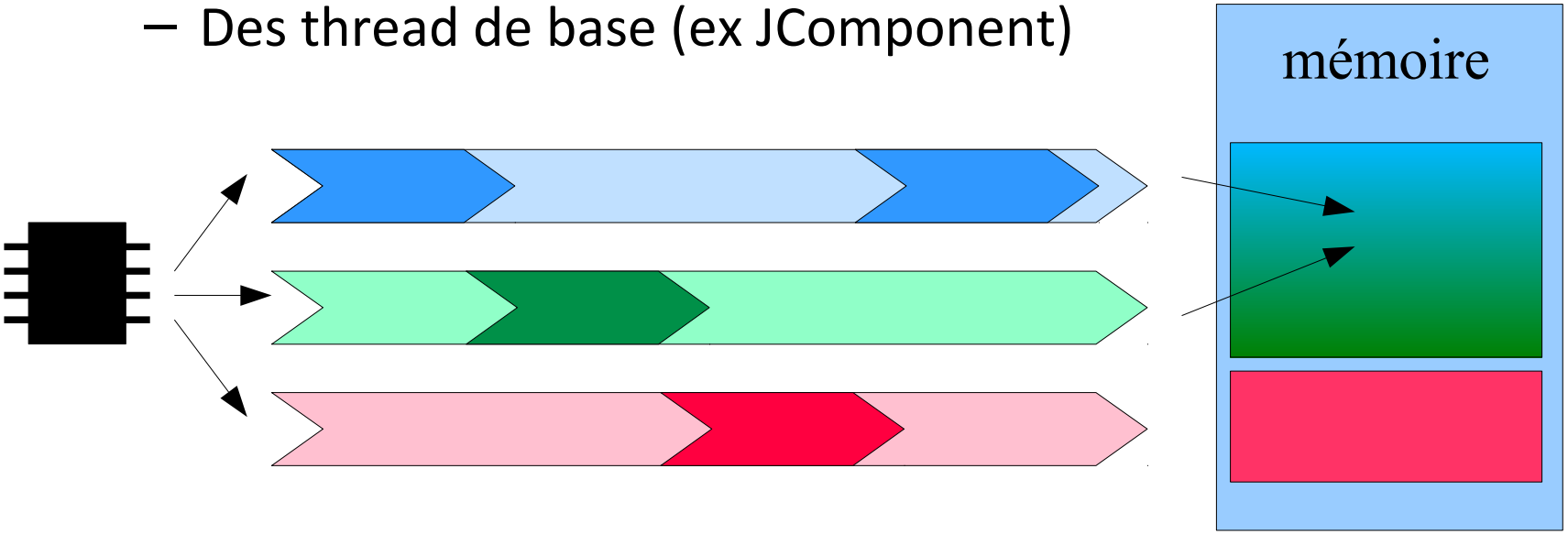
Thread

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



Thread

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



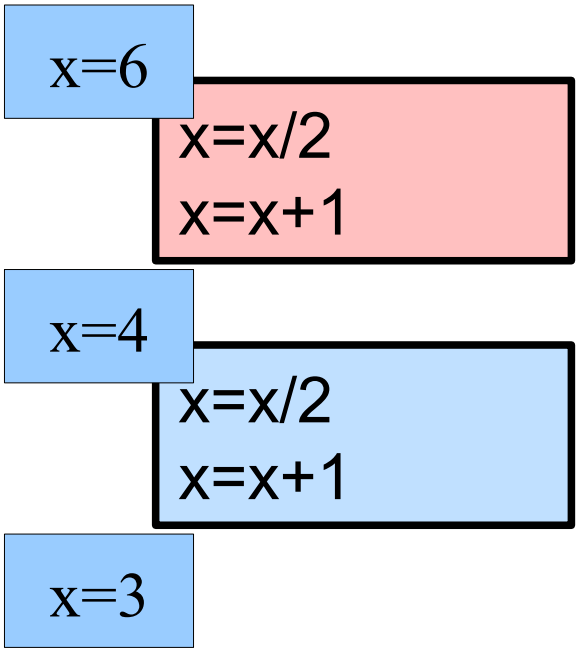
- Problèmes des threads
  - Méthodes ré-entrantes, Concurrence d'accès
  - Synchronisation, stop/pause

Thread pb - concurrence accès

- Variable commune
  - Modifier x en x/2+1

```
x=x/2
x=x+1
```

- Deux threads

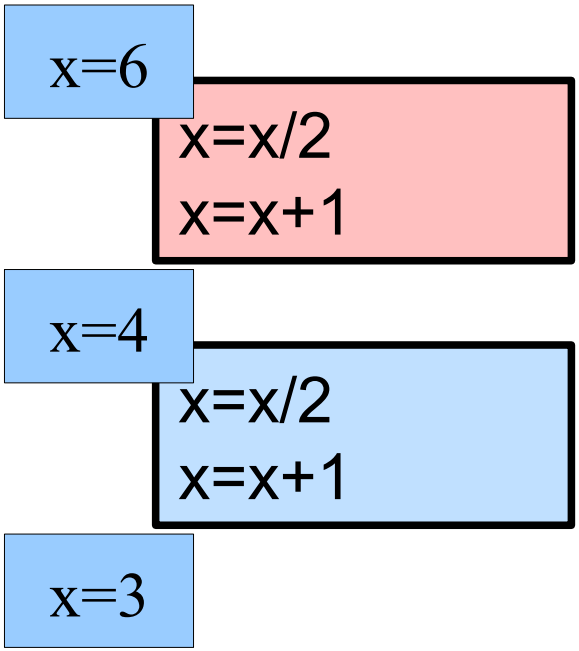


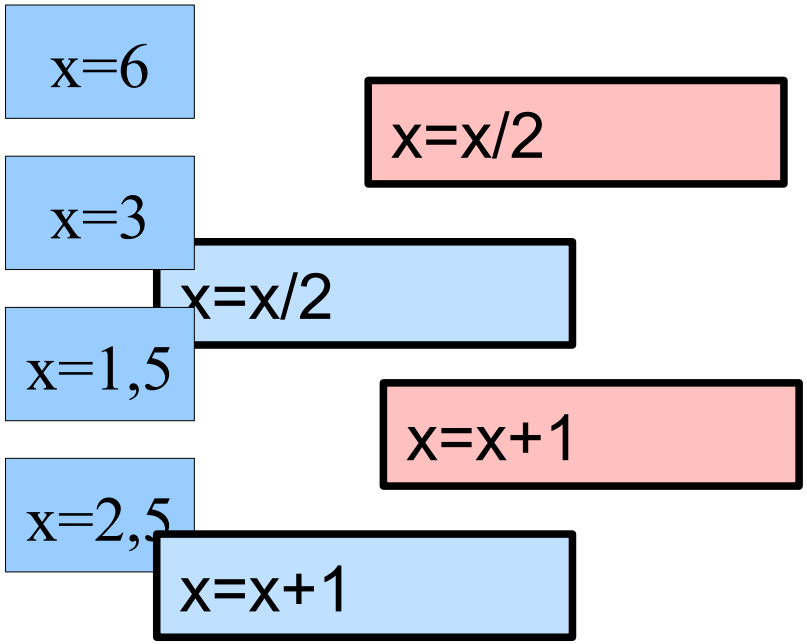
Thread pb - concurrence accès

- Variable commune
  - Modifier x en x/2+1

```
x=x/2
x=x+1
```

- Deux threads

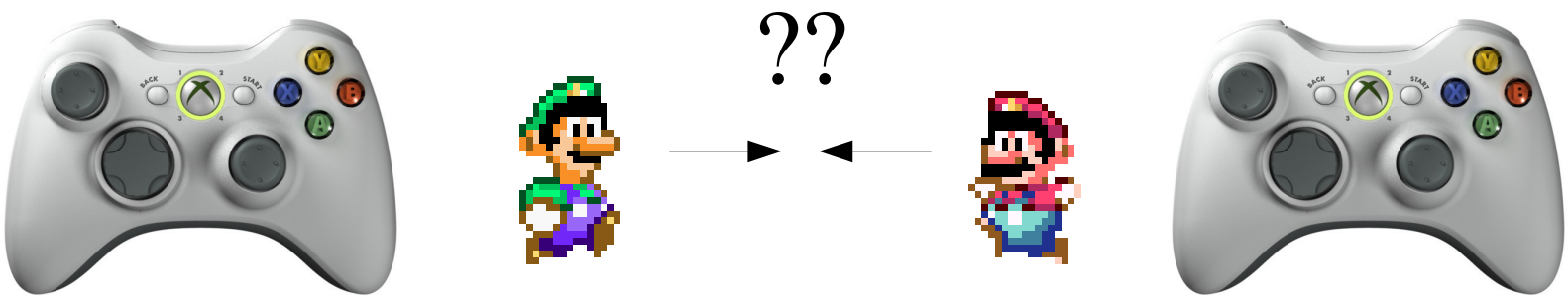




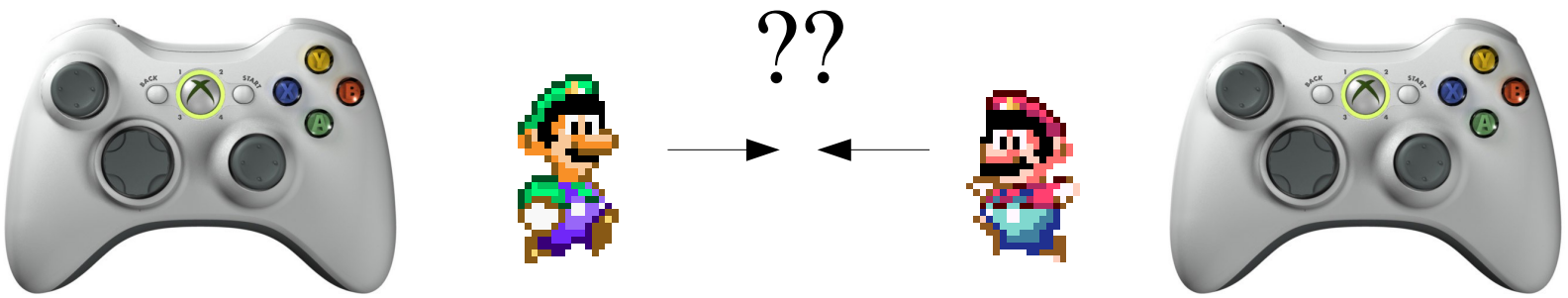
Thread

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

- Problèmes des threads
  - Méthodes réentrantes, Concurrence d'accès
  - Synchronisation, stop/pause



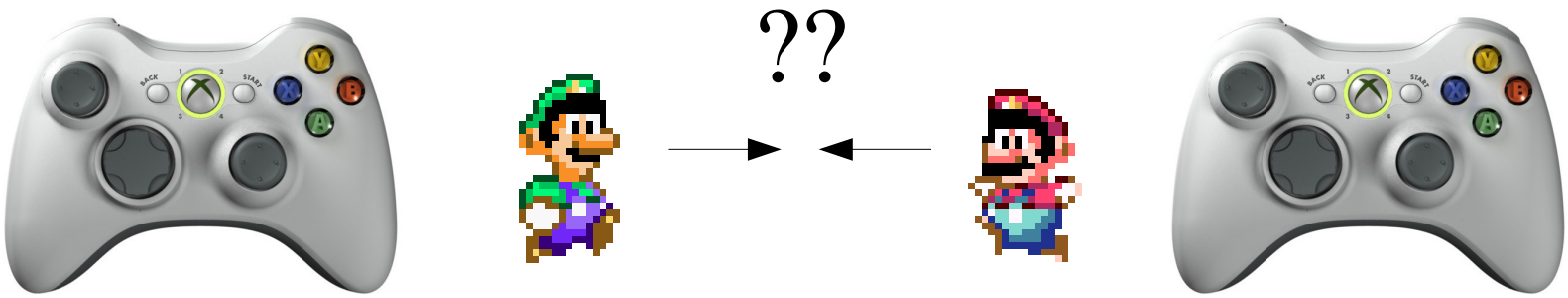
Thread




```
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
}
```

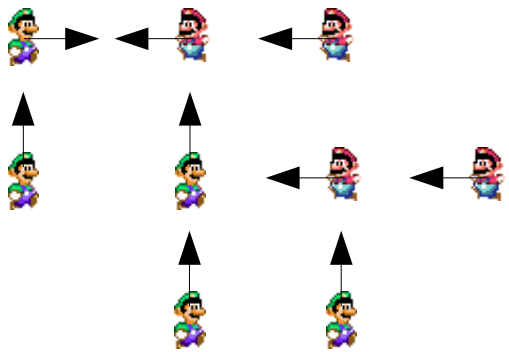
```
If ( p [ Luigi.x + 1 ] == libre)
{
    Luigi.x=Luigi.x+1

    p [ Luigi.x ] = occupé
}
```

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

Thread

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

- Problèmes des threads
  - Méthodes réentrantes, Concurrence d'accès
  - Synchronisation, stop/pause



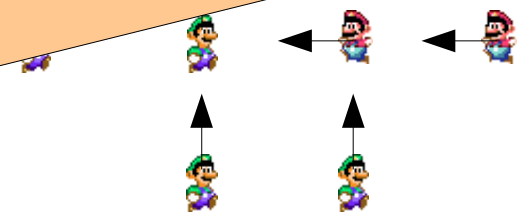
Thread

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

- Problèmes des threads
  - Méthodes ré

On va éviter les threads

Centraliser au maximum
Contrôler au mieux



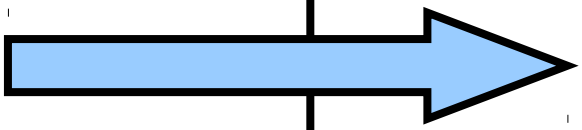
Boucle principale

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

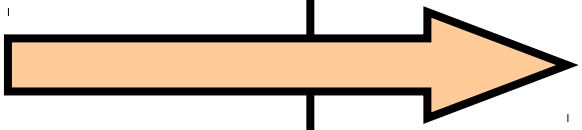
- Classe Principale
  - Boucle boolean fini (gagne/perdu)
  - Appels méthodes de Game
    - Méthode update
    - Méthode render

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();
  }

}
```

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();
  }

}
```

Classe Principale

```
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

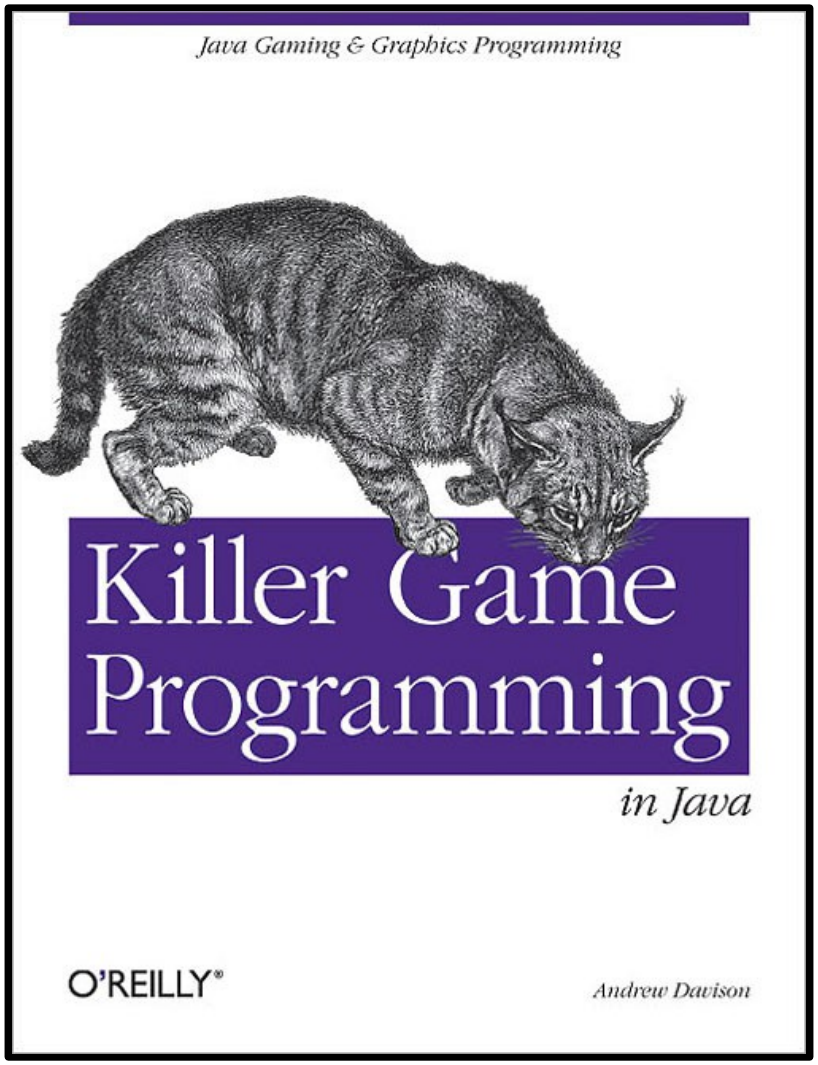
Démonstration Partie 1

Boucle et affichage

Plan

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

Différentes boucles principales



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

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

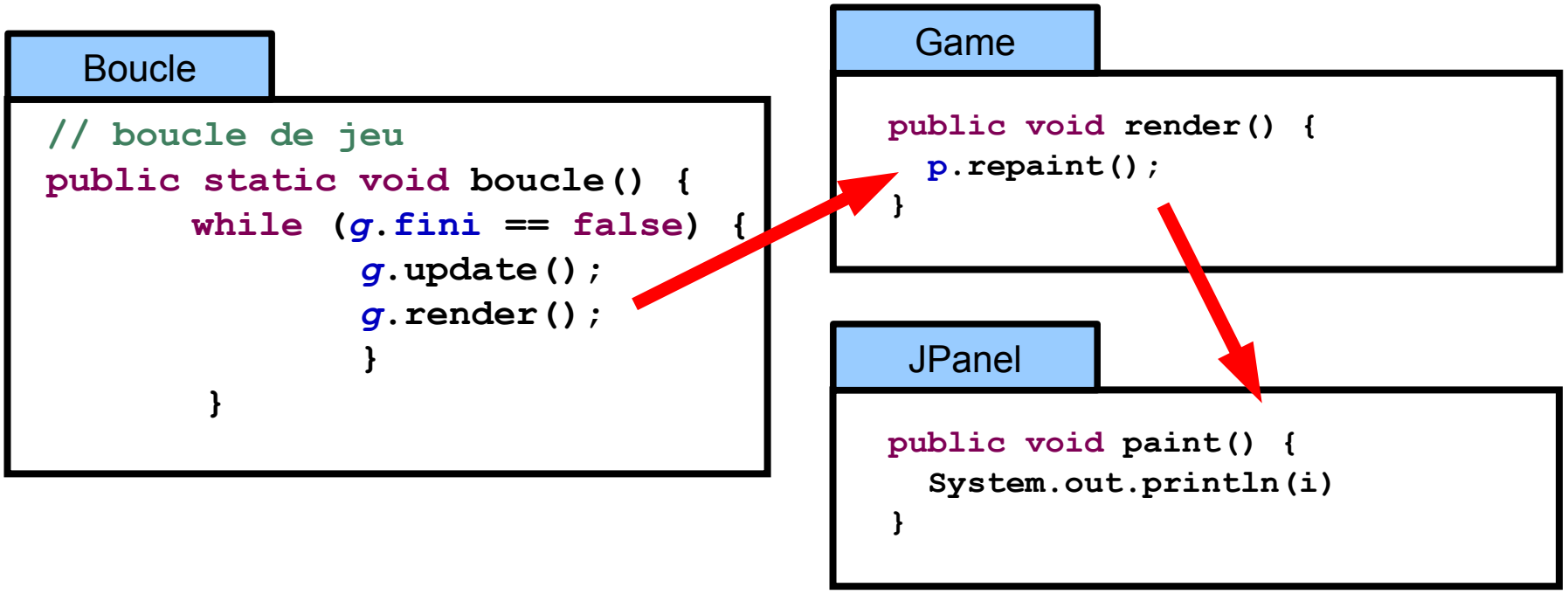
```
public void paint() {
  System.out.println(i)
}
```

- 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

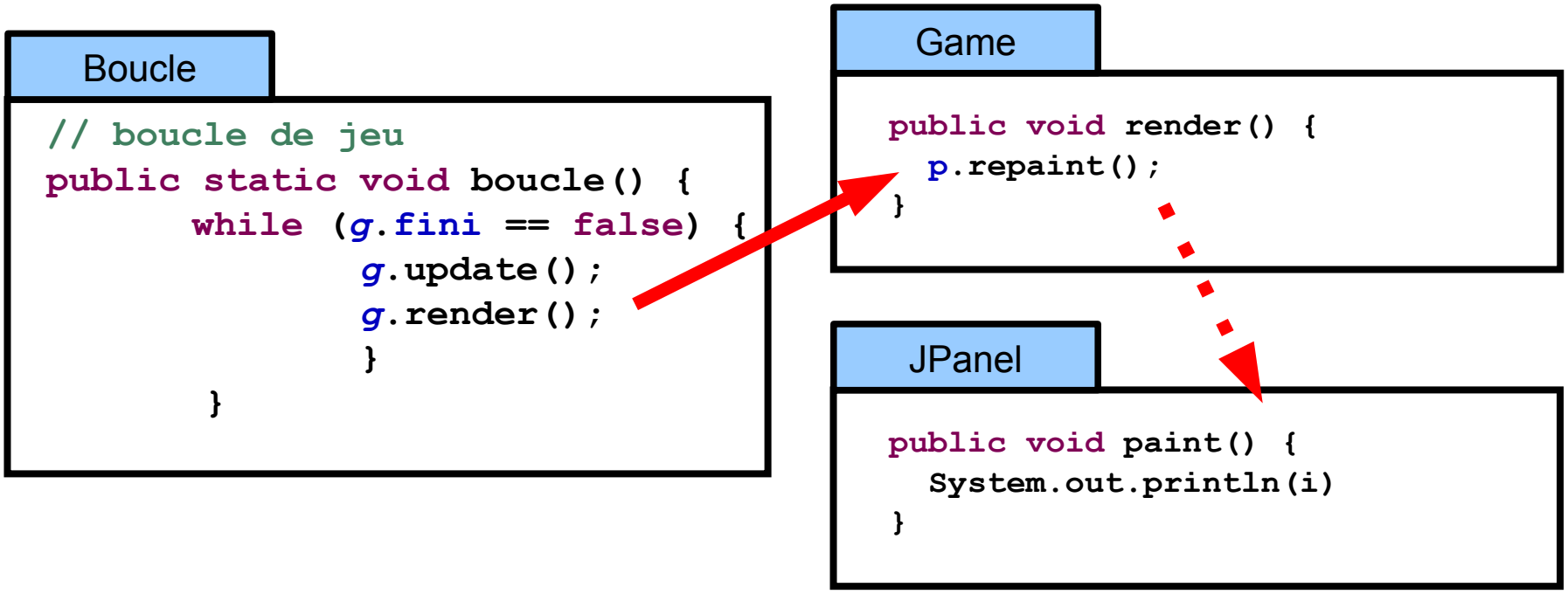


- Problème

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

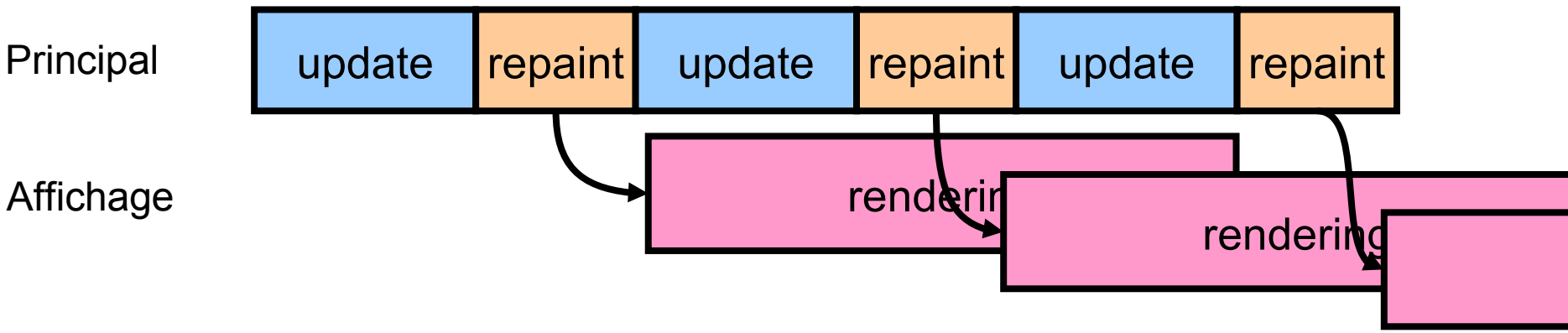


- Problème

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

Boucle simple (2)

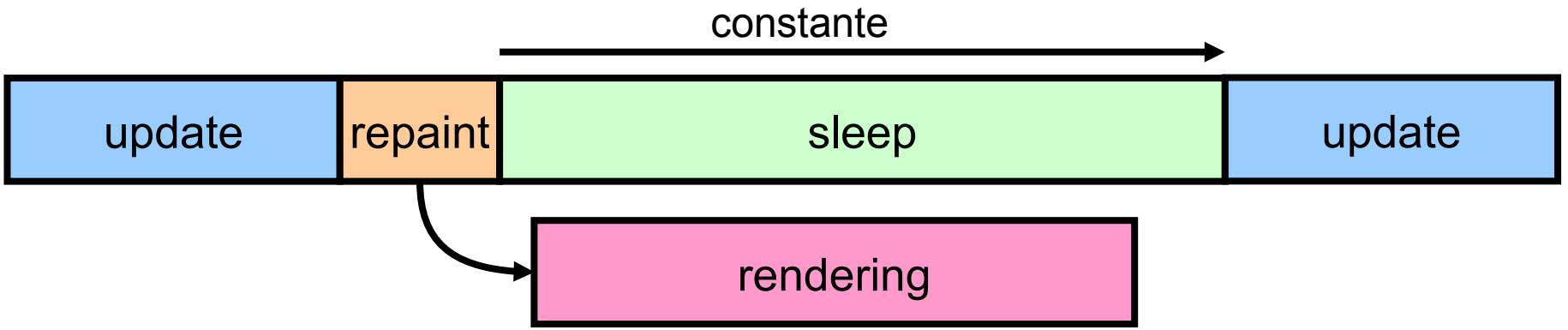
- Origine du problème
  - Repaint transmet la **demande** d'affichage
  - Repaint fusionne la demande



- Solution
  - Pause
  - Rendre la main à l'affichage

Boucle + sleep

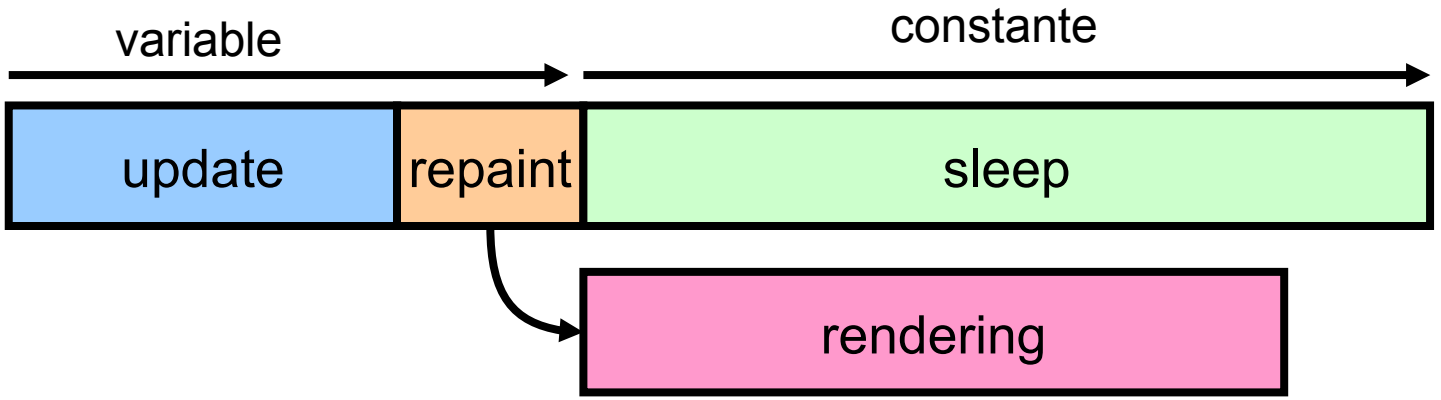
- Redonner la main à l'affichage (Méthode sleep)

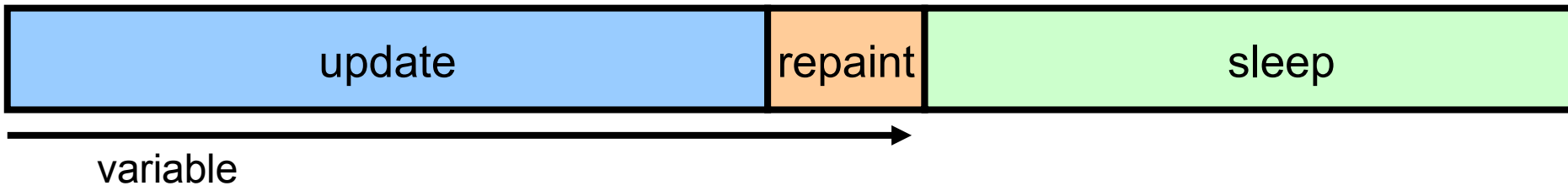



```
static void boucle() throws InterruptedException {

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

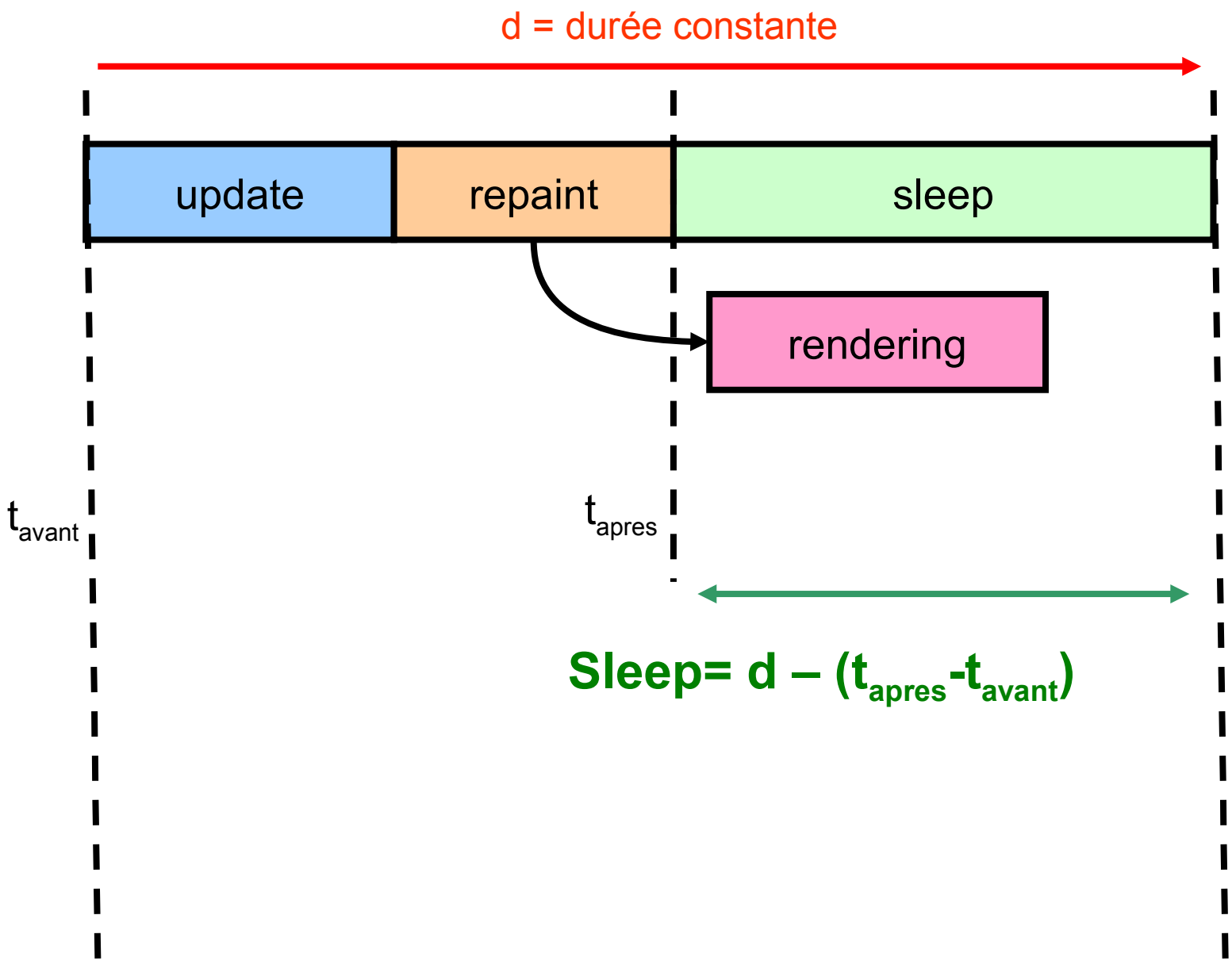
Boucle + sleep (2)

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

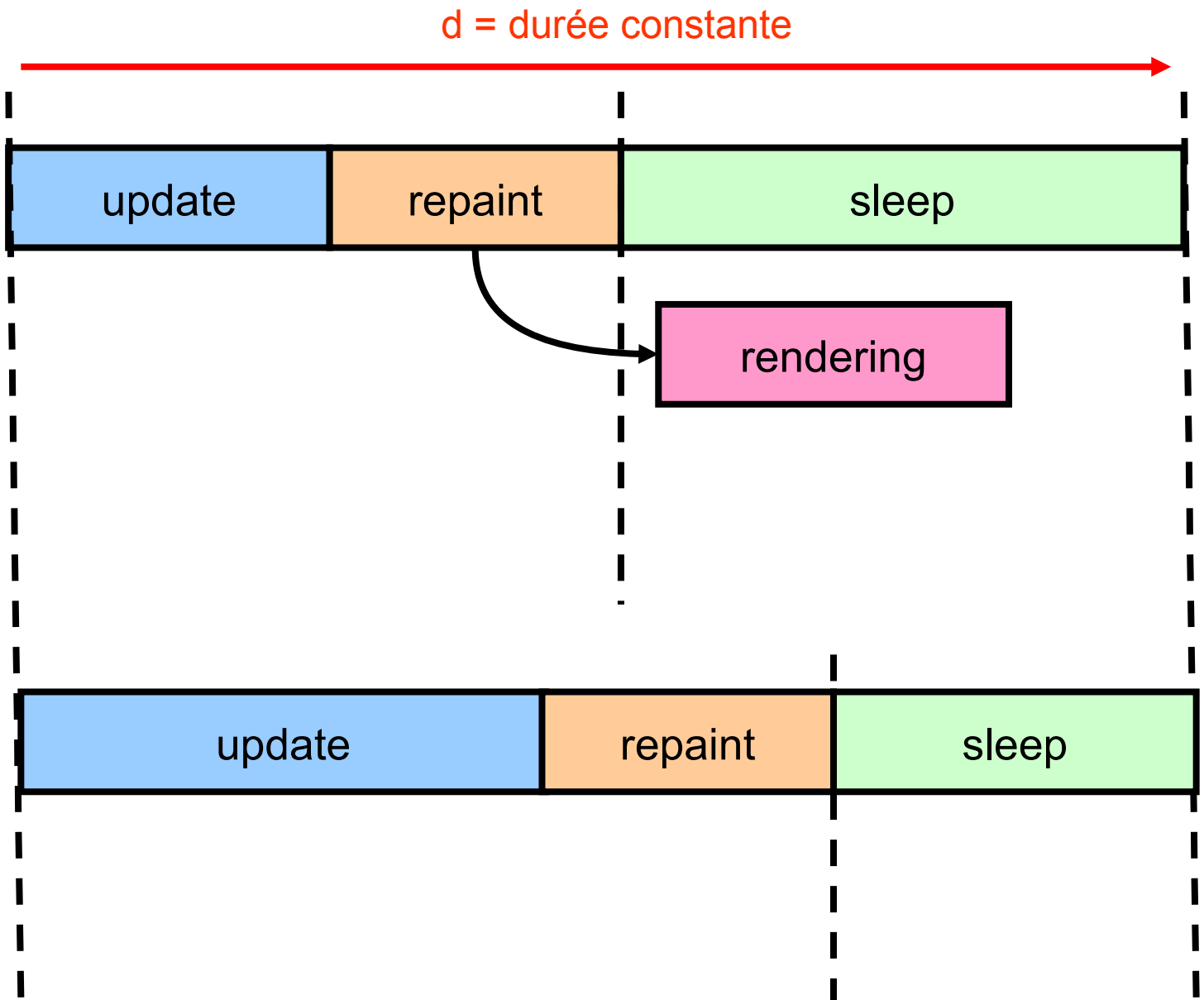




Boucle + sleep adaptatif



Boucle + sleep adaptatif (2)



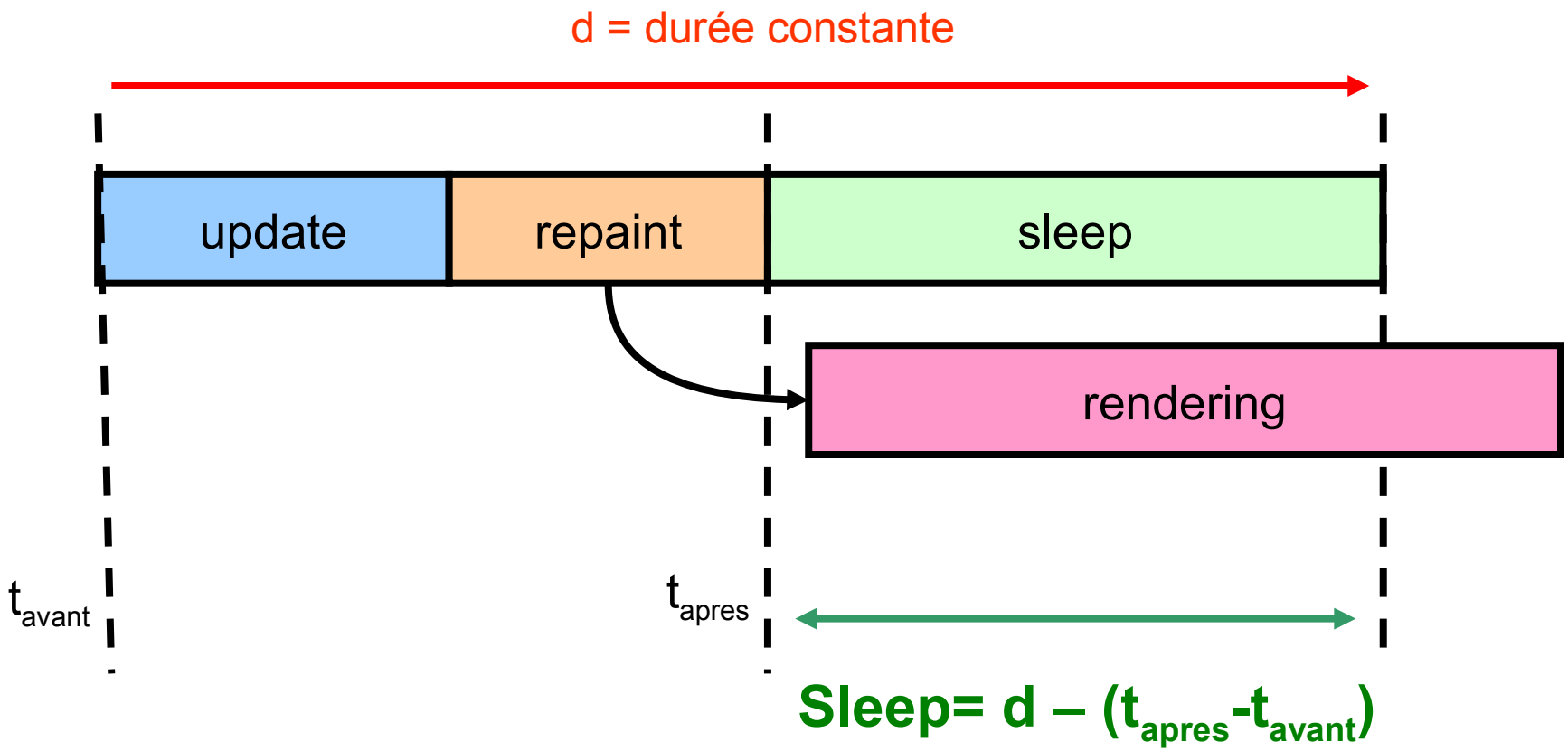
Boucle + sleep adaptatif (3)

- 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 aprés
                long apres = System.currentTimeMillis();

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

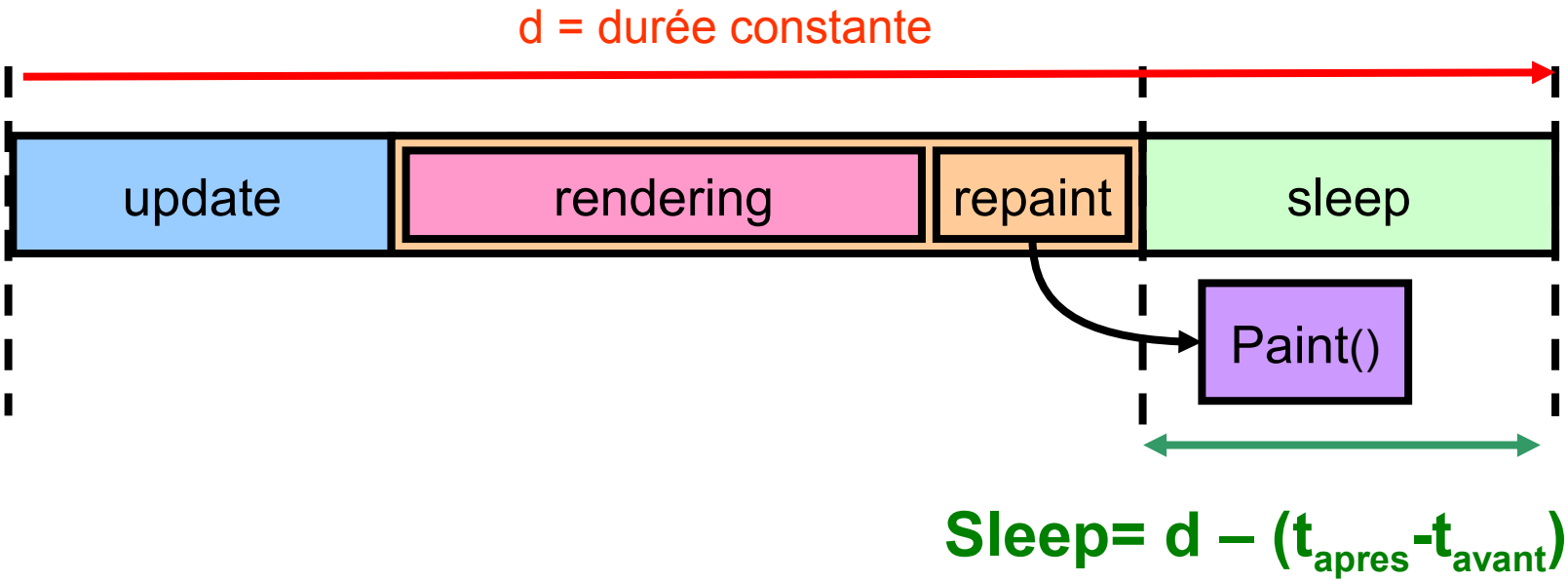
Boucle + sleep adaptatif (4)

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



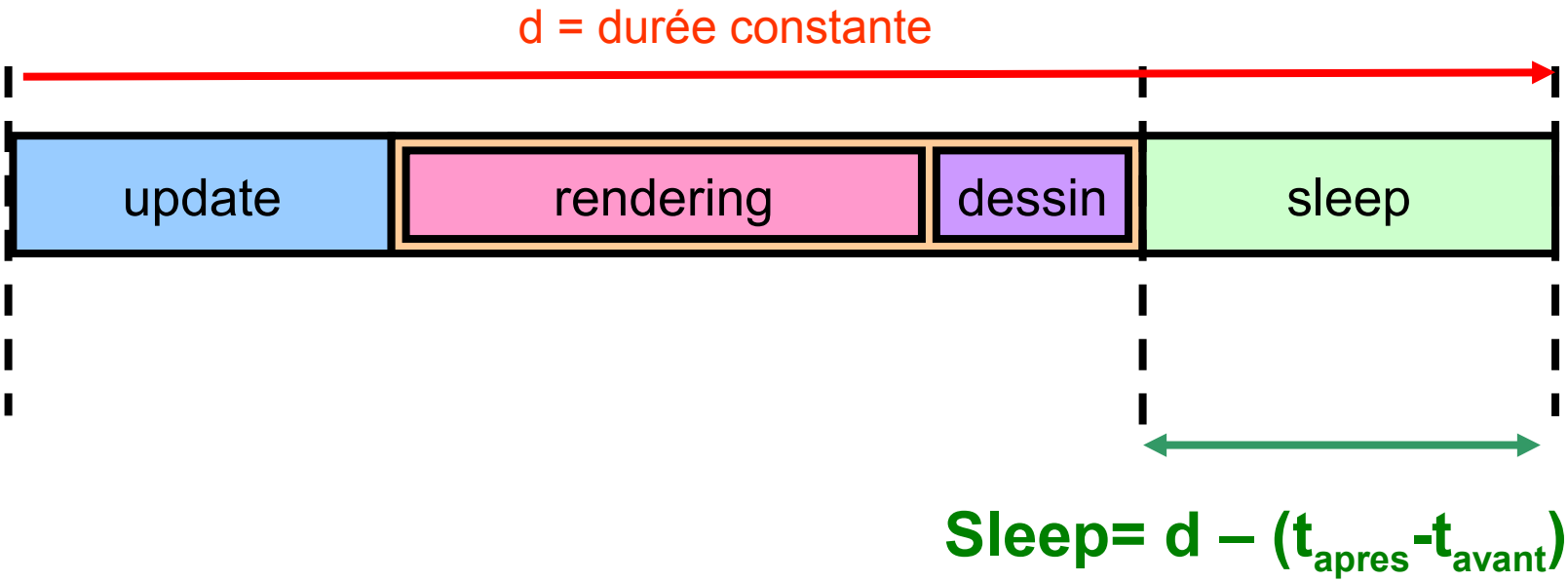
Active rendering & double buffering

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



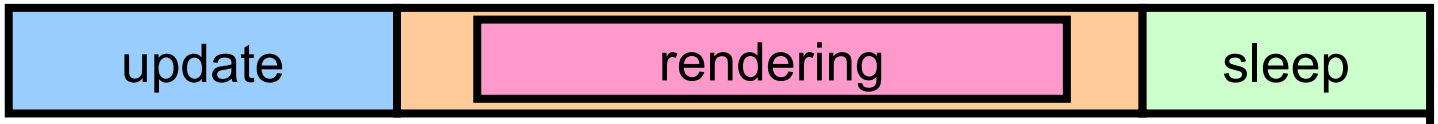
Active rendering & double buffering

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

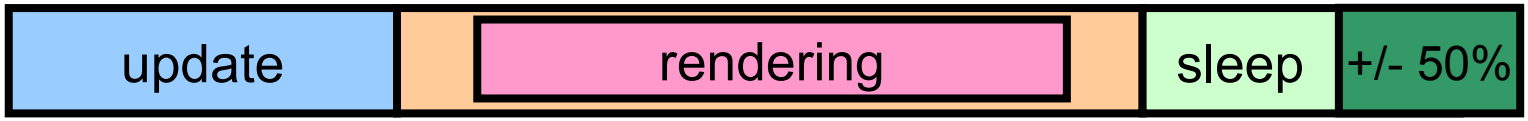


Active rendering & double buffering (2)

Cas idéal

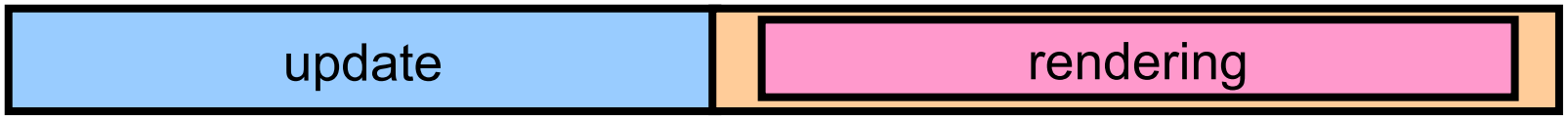


Pb1 – précision du sleep



Pb2 – précision du timer

Pb3 – pas assez de temps disponible



FPS et probleme de temps

- 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

Pb1: Précision du Timer

- 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

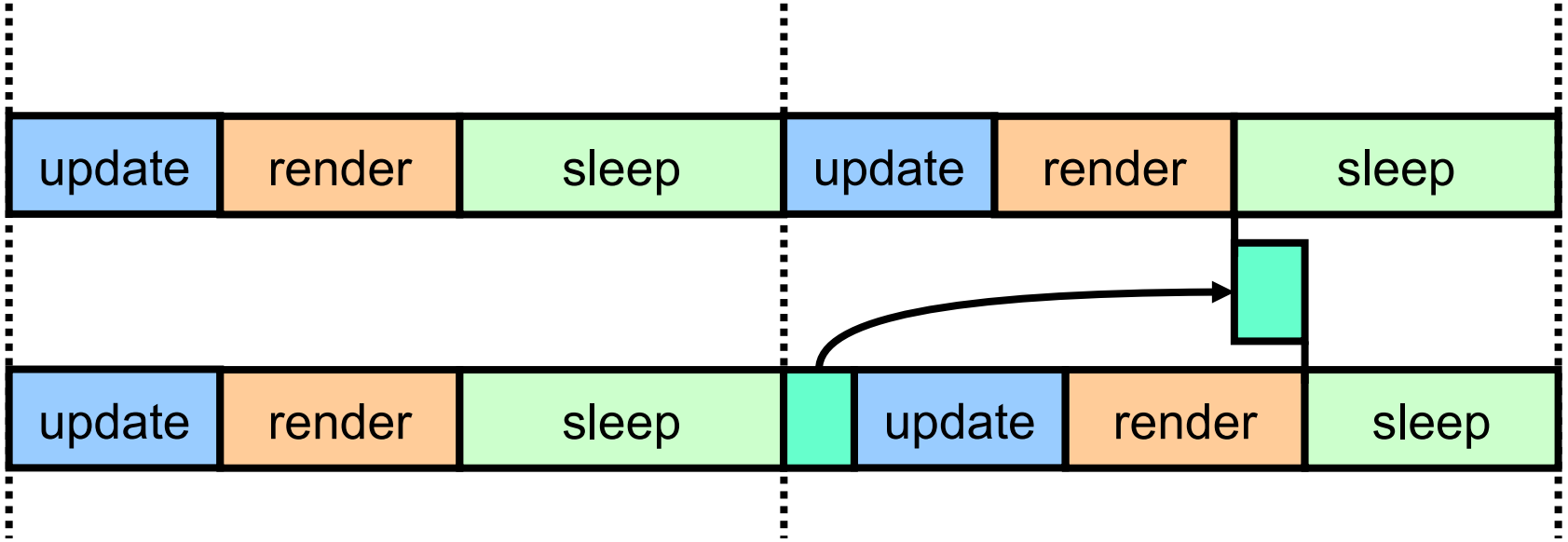
- Le timer ne peut pas distinguer 0 de 15 ms
  - Dans tous les cas, le temps évalué sera de 0
  - Le sleep sera tout le temps de durée (ex 10ms)
  - L'itération durera entre 10 et 25 ms

- Solution
  - Utiliser la classe **Perf**

Pb2: Précision du sleep

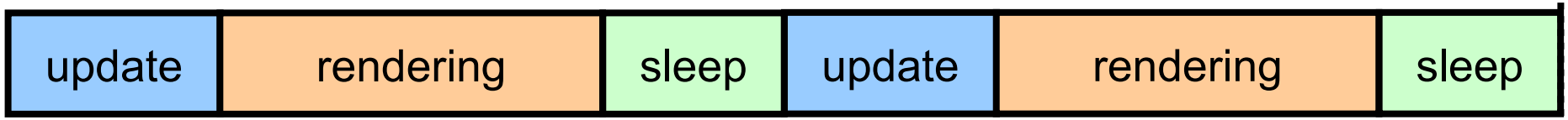
- La méthode sleep arrète le processus
  - Pendant un certain temps
  - Mais avec des erreurs (de 1% à 20%)

- Retenir le delai et le prendre en compte



Pb3: Pas assez de temps de calcul

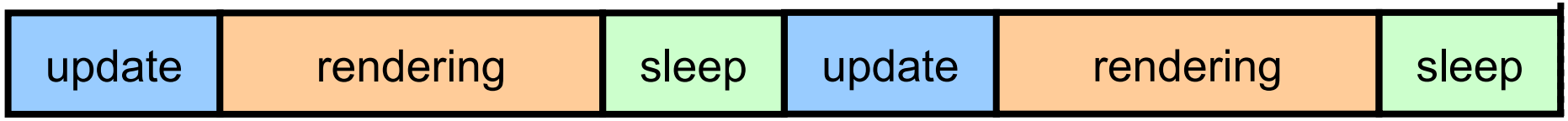
- Cas normal



- Cas problématique

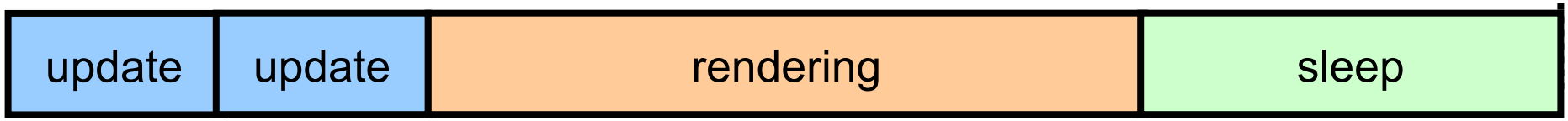
Pb3: Pas assez de temps de calcul

- Cas normal



- Cas problématique

- Solution
  - Faire plusieurs update pour un affichage
  - Le jeu reste aussi rapide mais moins fluide
    - Évaluer le décalage et rattraper
    - Dépend des durée de l'update et de l'affichage



Démonstration Partie 2

Gestion du temps

Sauf que …

- L'année dernière (pour les présents)
  - Jeu n'était pas fluide

Sauf que …

- L'année dernière (pour les présents)
  - Jeu n'était pas fluide
  - Changement de machine ralentit

Sauf que …

- L'année dernière (pour les présents)
  - Jeu n'était pas fluide
  - Changement de machine ralentit

- Expériences

Sauf que …

- Classe Thread
  - Pas 20 % Mais 100 %
  - À la place de 9ms
    - 5 ms ou 17 ms

Sauf que …

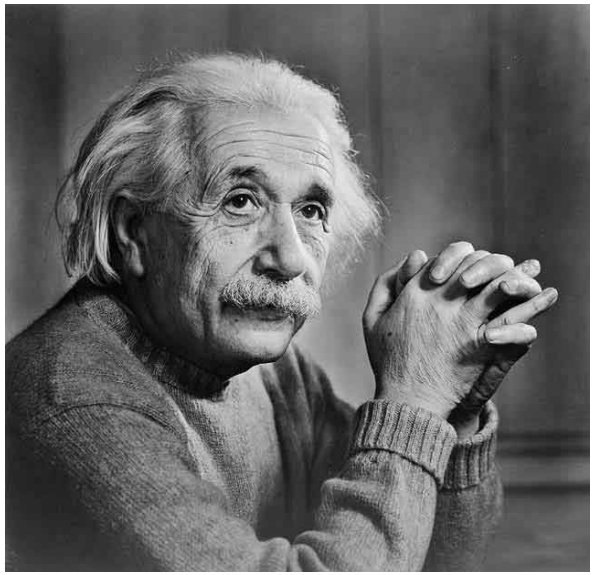
- Classe Thread
  - Pas 20 % Mais 100 %
  - À la place de 9ms
    - 5 ms ou 17 ms

- Si on est à 17 ms
  - 58 Fps

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

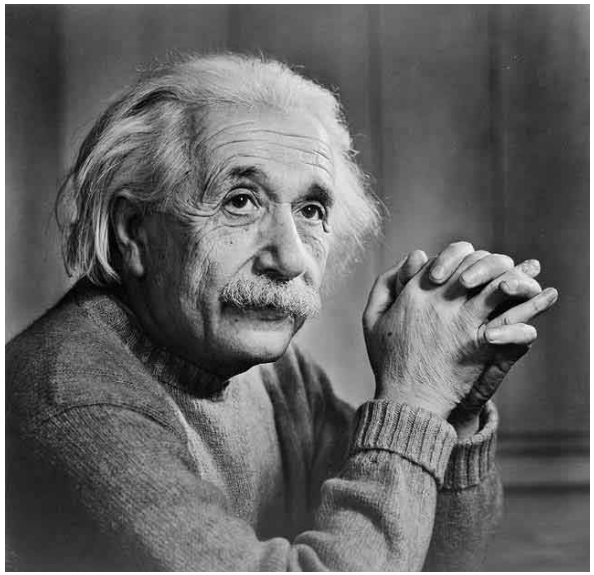
Sauf que …

- Retordre le temps ?



Sauf que …

- Retordre le temps ?
- Méthode System.nanoTime
  - *« This method provides nanosecond precision, but not necessarily nanosecond resolution »*
- En croisant les doigts



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");

}
```

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");

}
```

Moralité (1)

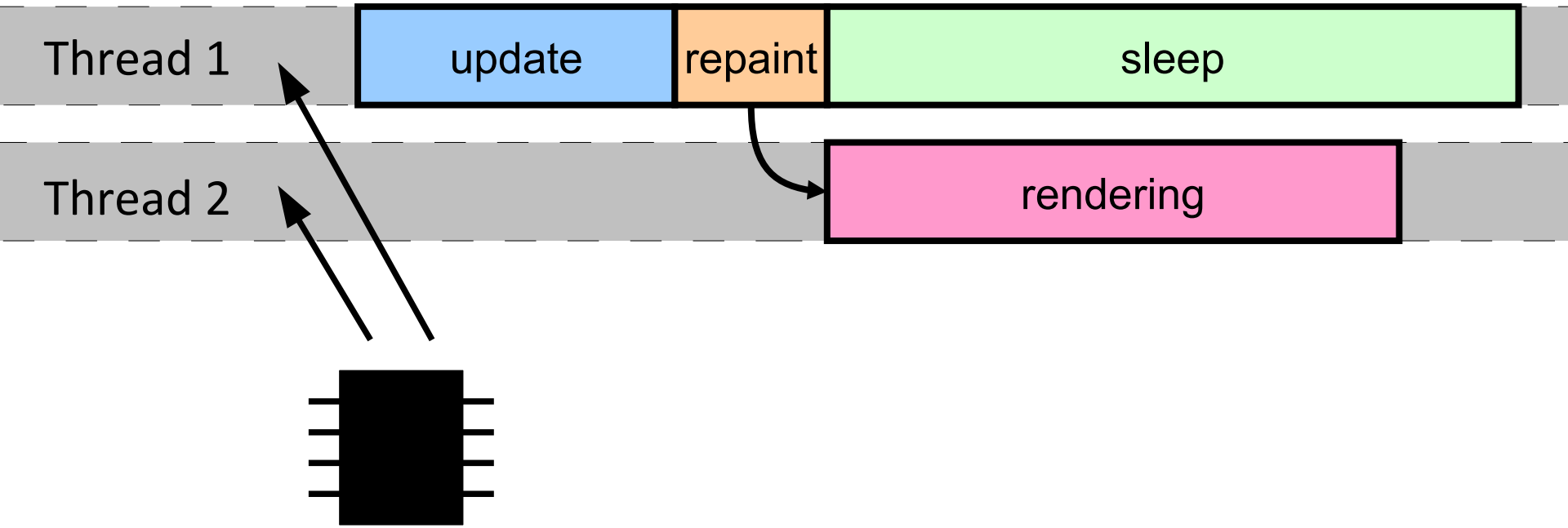
1. Ne partagez pas votre temps processeur

Moralité (1)

1. Ne partagez pas votre temps processeur
2. faites une boucle simple

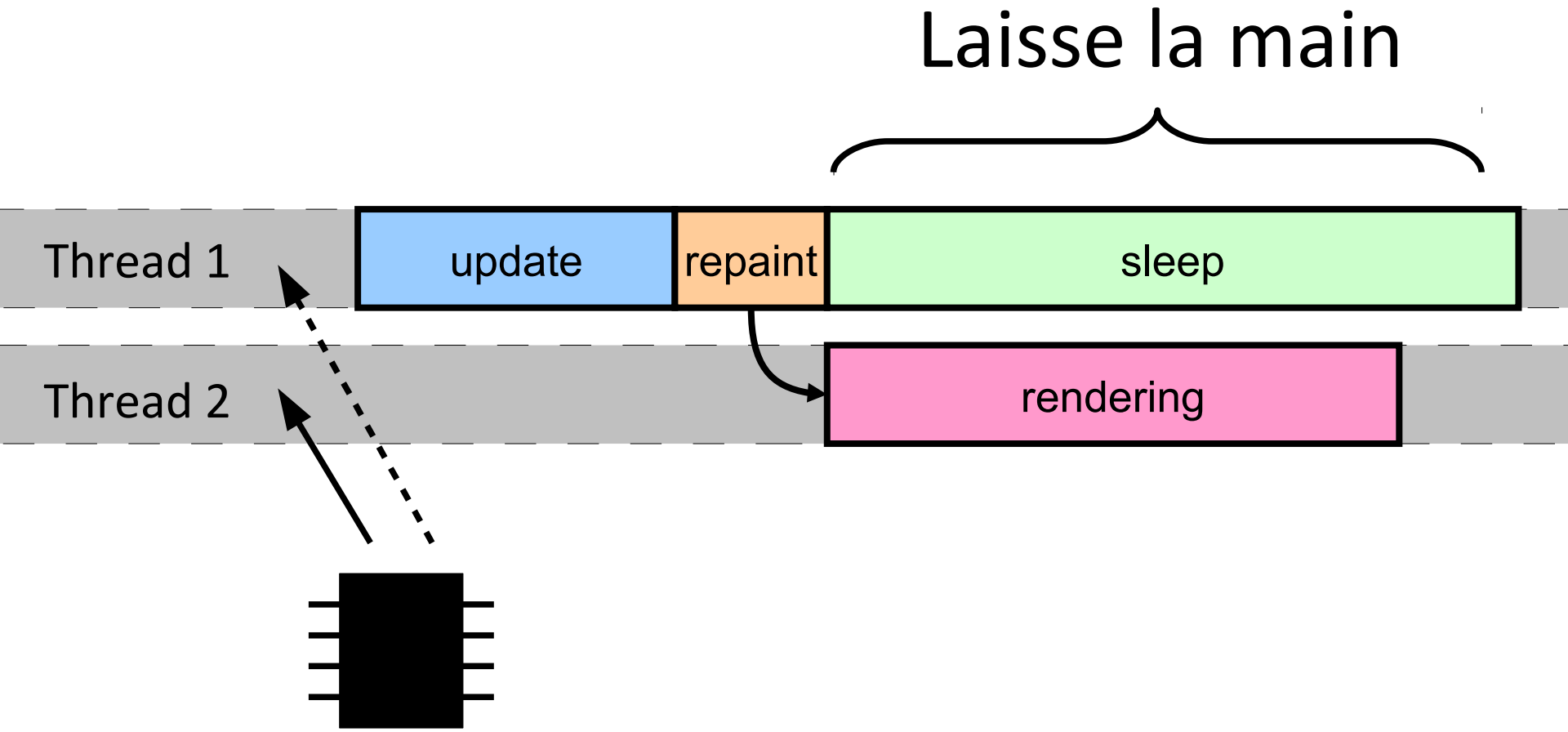
Moralité (2)

- Avant



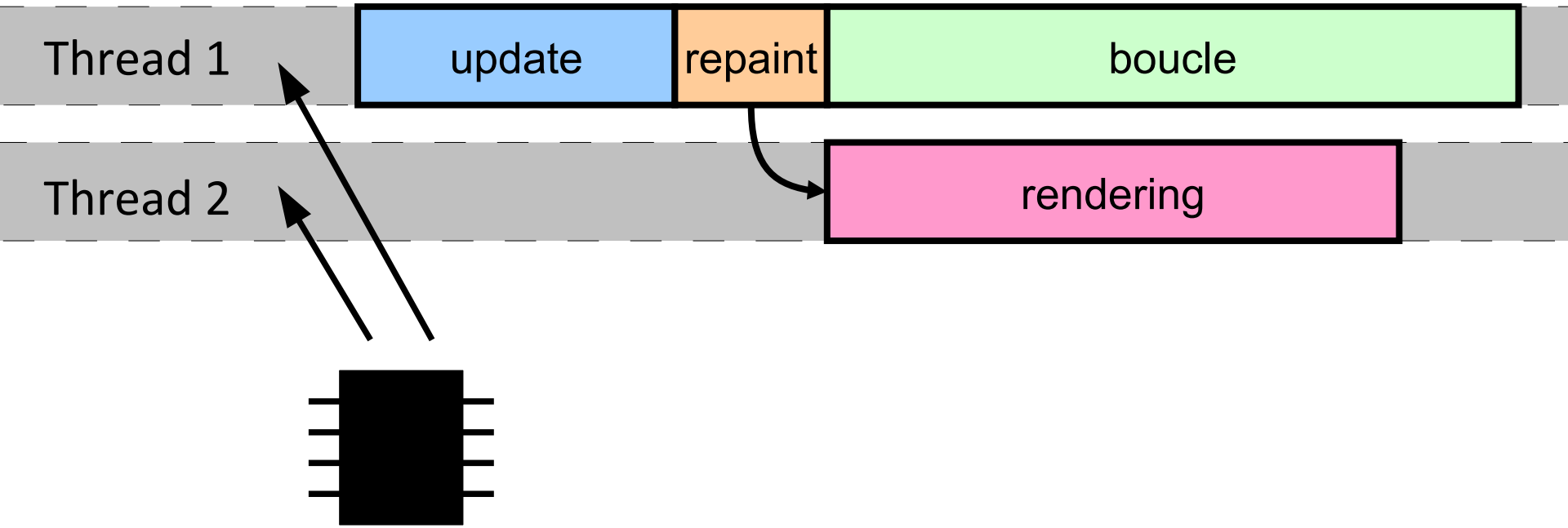
Moralité (2)

- Avant



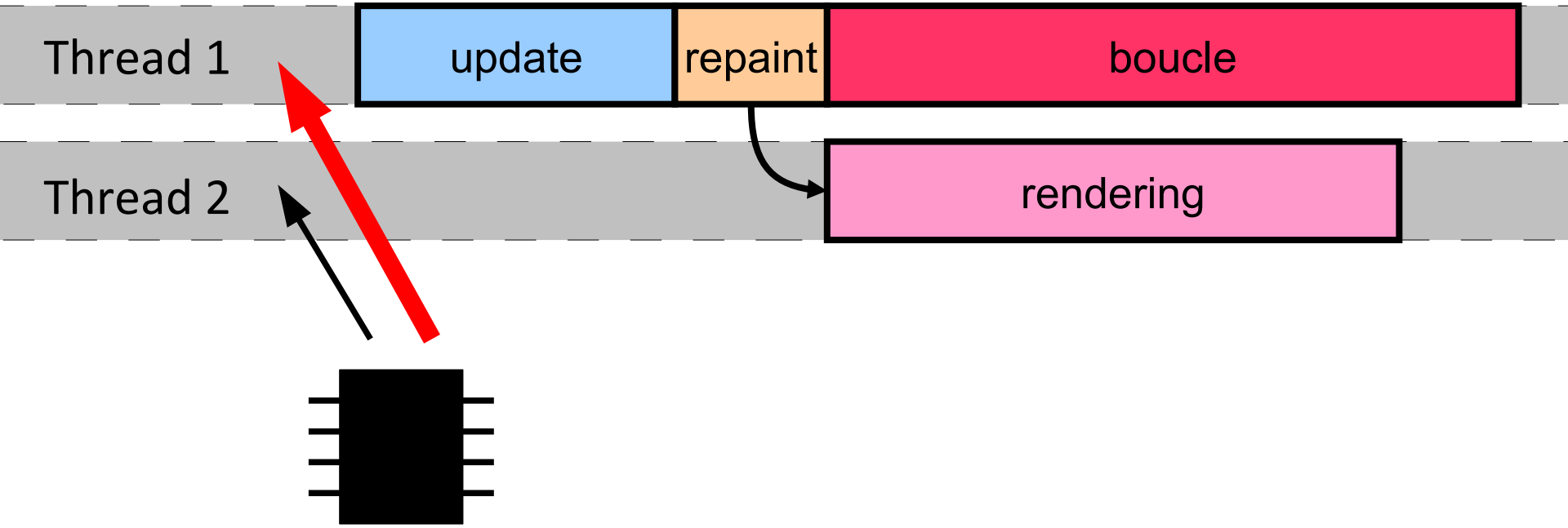
Moralité (2)

- Maintenant



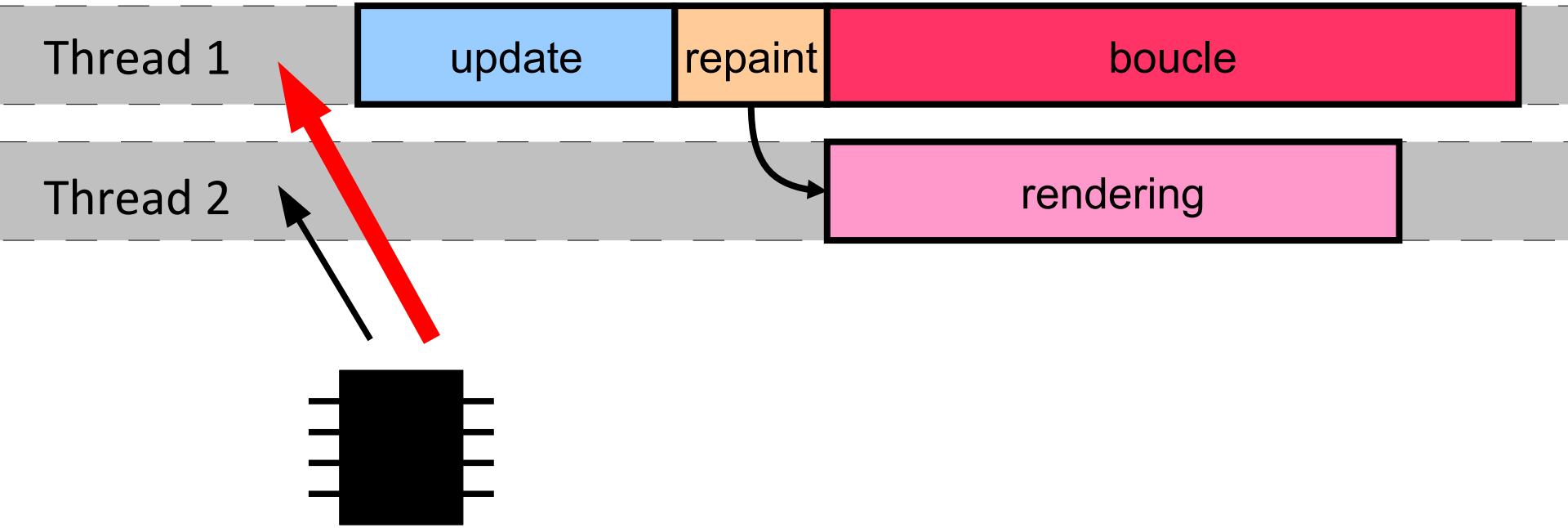
Moralité (2)

- Maintenant



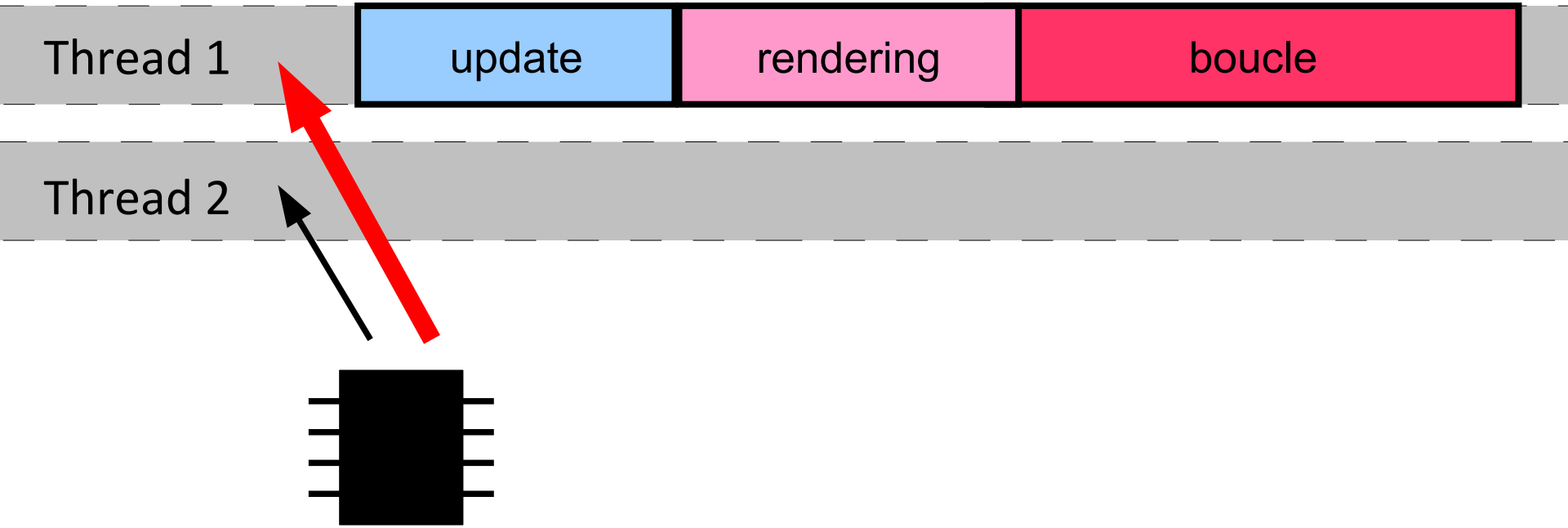
Moralité (2)

- Active rendering
  - Rendu dans boucle



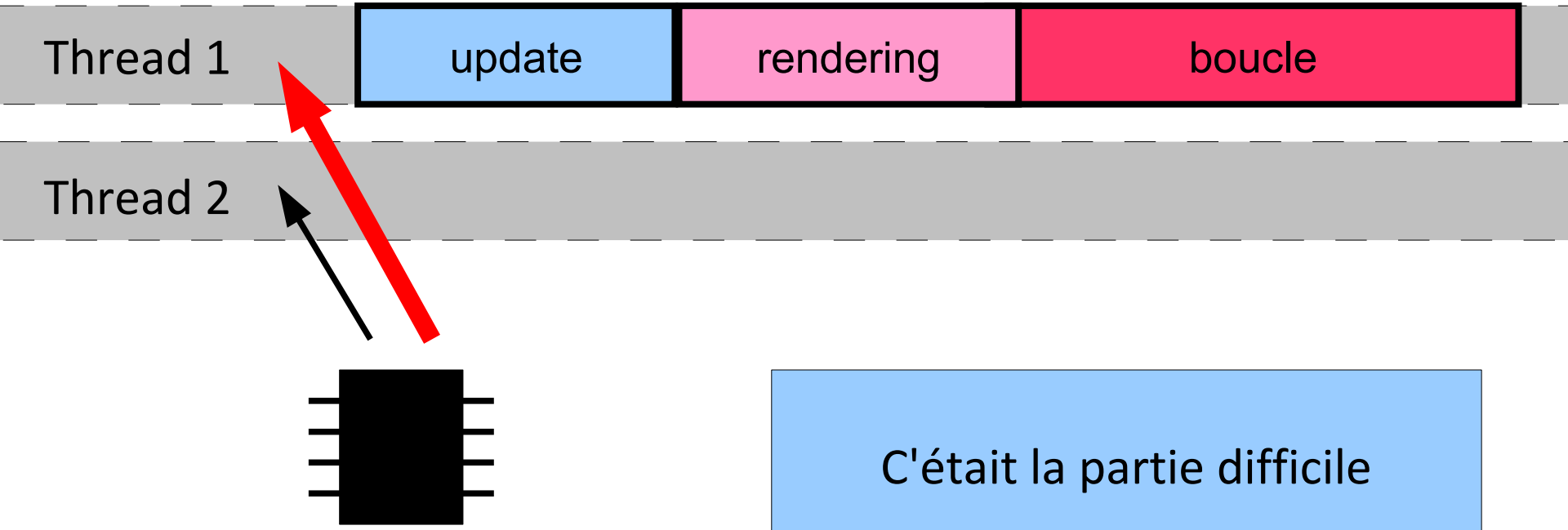
Moralité (2)

- Active rendering
  - Rendu dans boucle



Moralité (2)

- Active rendering
  - Rendu dans boucle



C'était la partie difficile

Bilan

- 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


```
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 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

Exemple de jeu

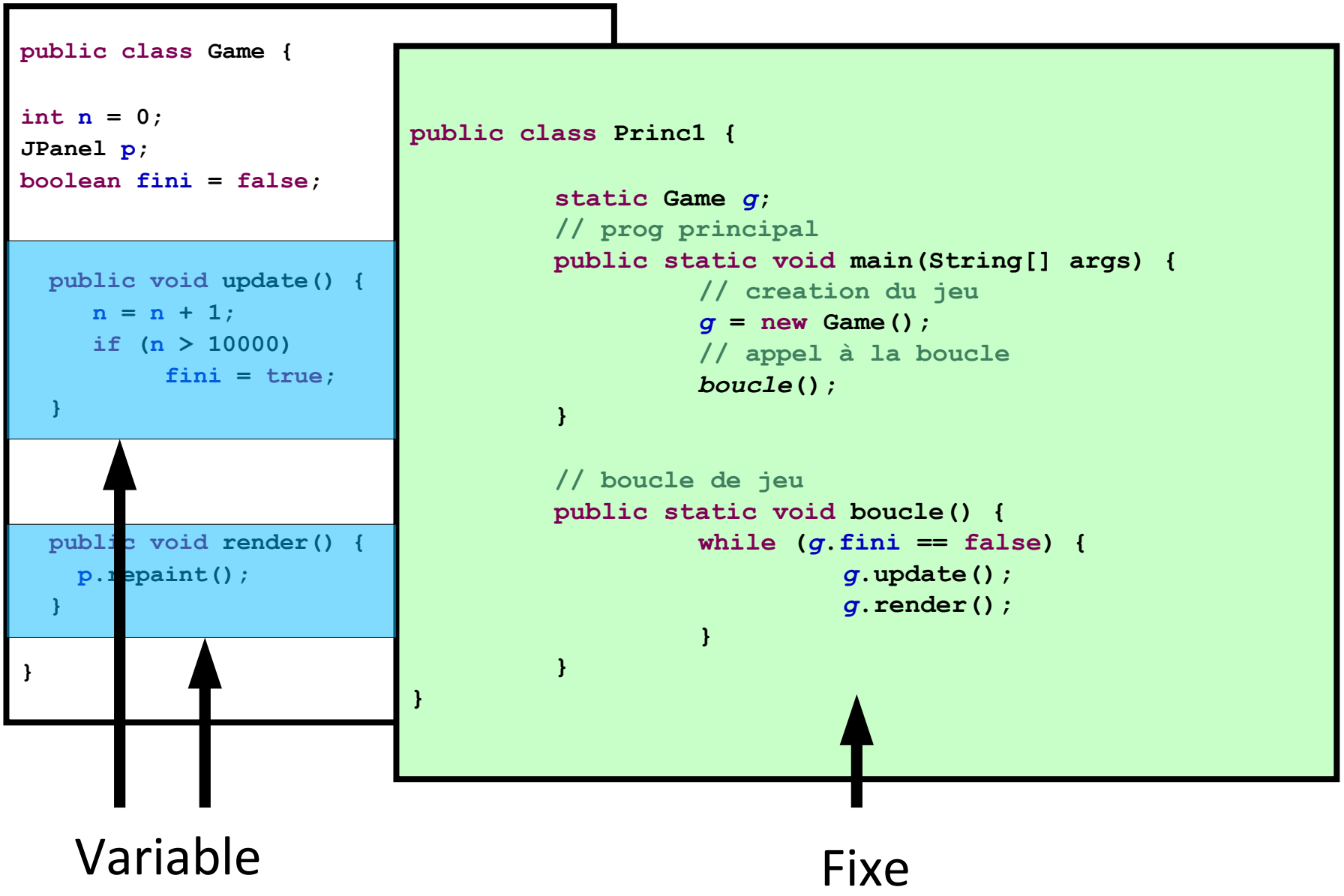

```
public class Game {

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


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


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

}
```

```
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();
                }
        }
}
```

1. pas de repaint

2. pas de thread

Exemple de jeu



Plan

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

Quoi mettre dans update ?

- Update()
  - Lié au Modèle

- Mise à jour
  - Lois du monde
  - Collisions
  - Contrôleurs du joueur
  - Intelligence artificielle

Lois du monde

- Règles du jeu
  - A définir

- Une itération 10ms

  Gestion des temps de déplacements

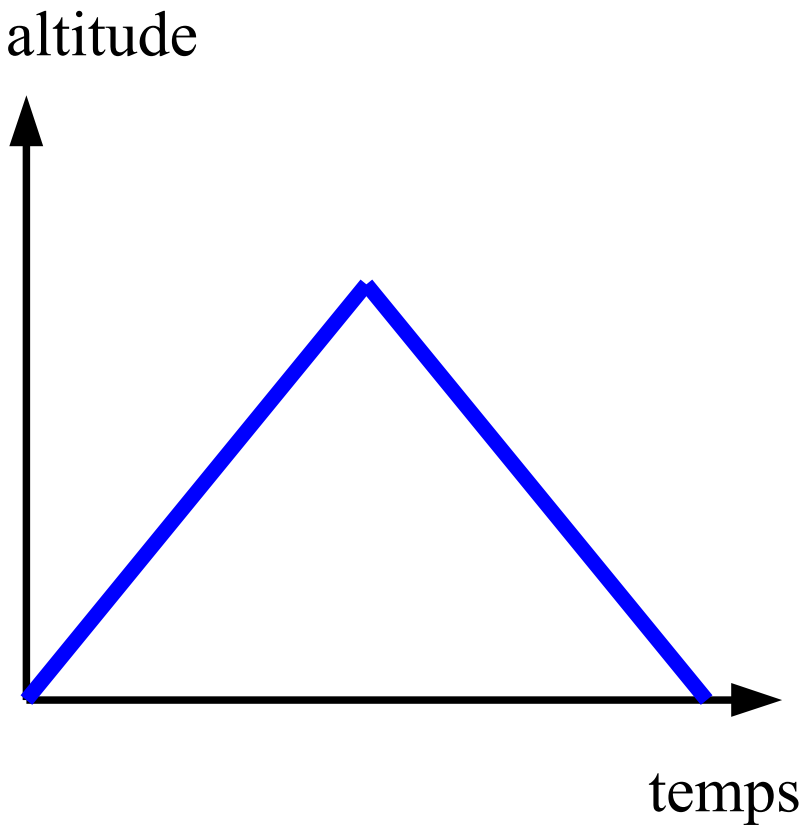
  1 pixel => 100 pixels / secondes

- Ressenti du joueur

  Exemple jeu de course

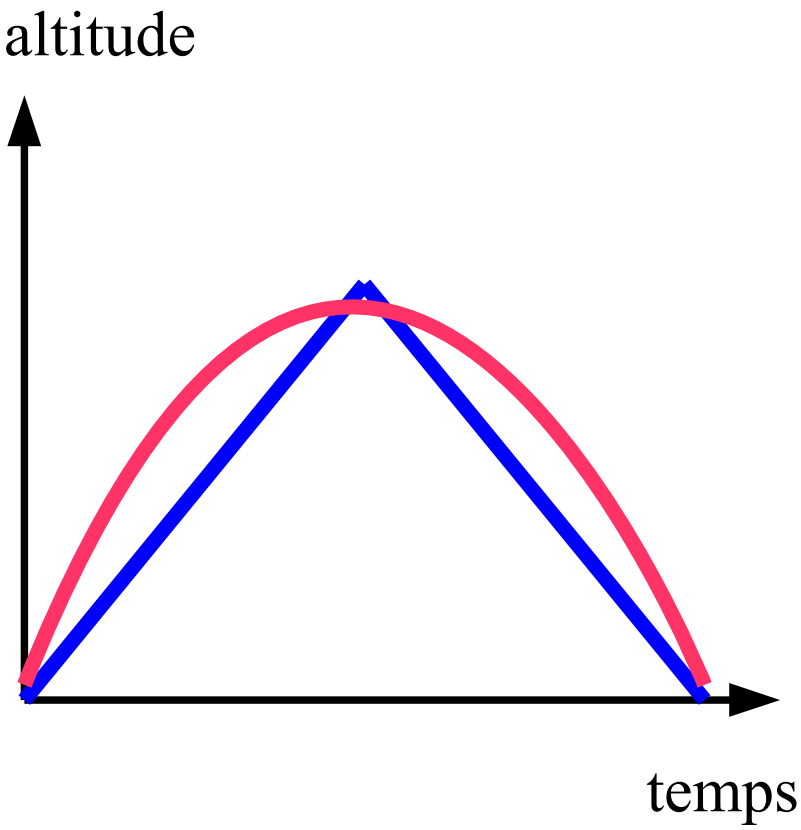
Lois du monde (2)

- Comportement de saut



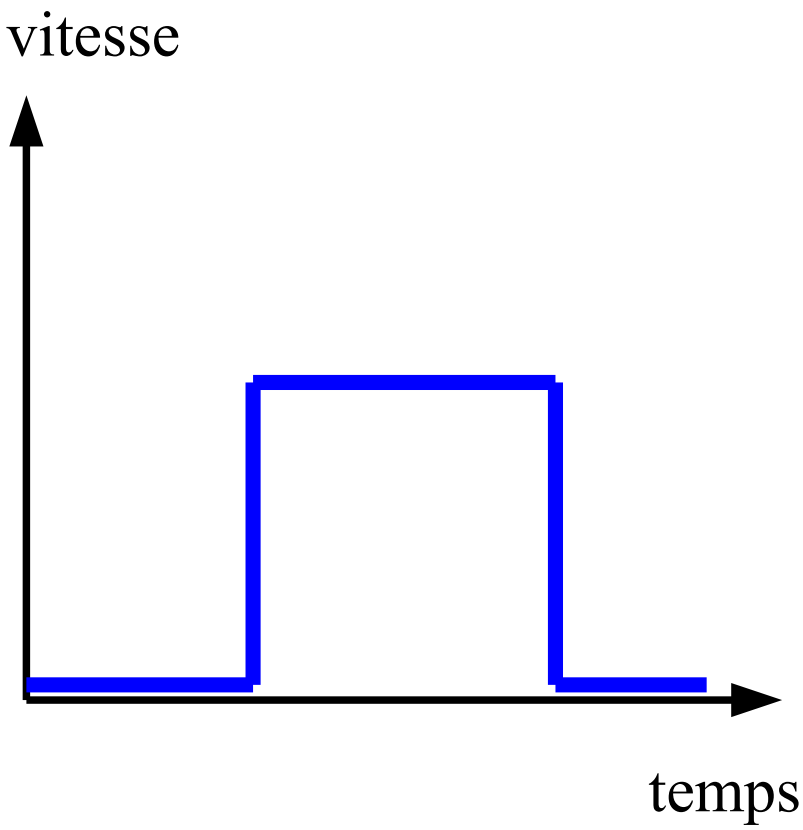
Lois du monde (2)

- Comportement de saut



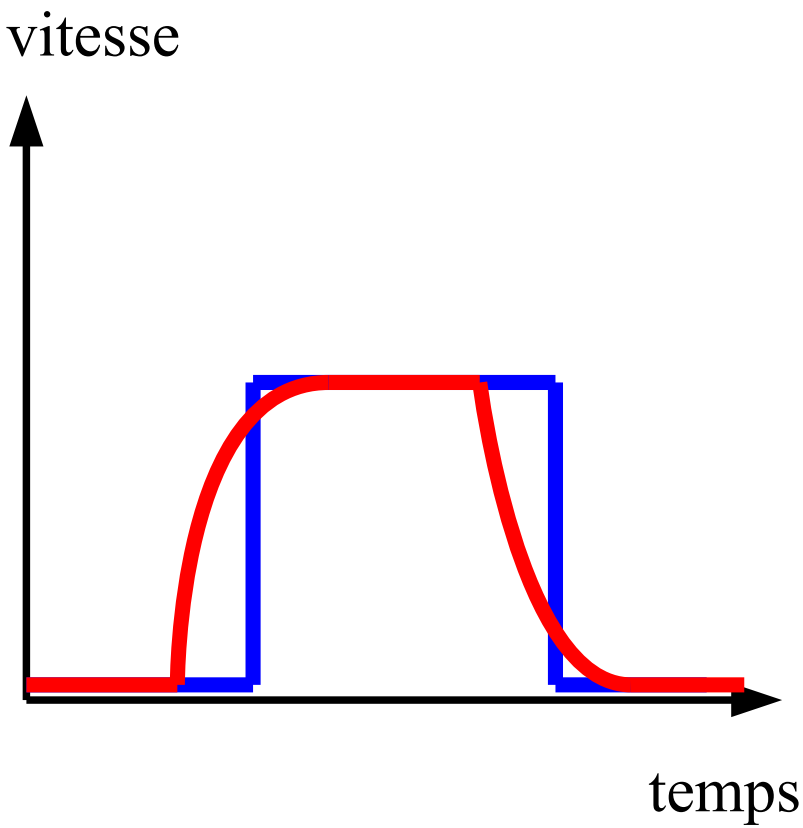
Loi du monde (3)

- Course



Loi du monde (3)

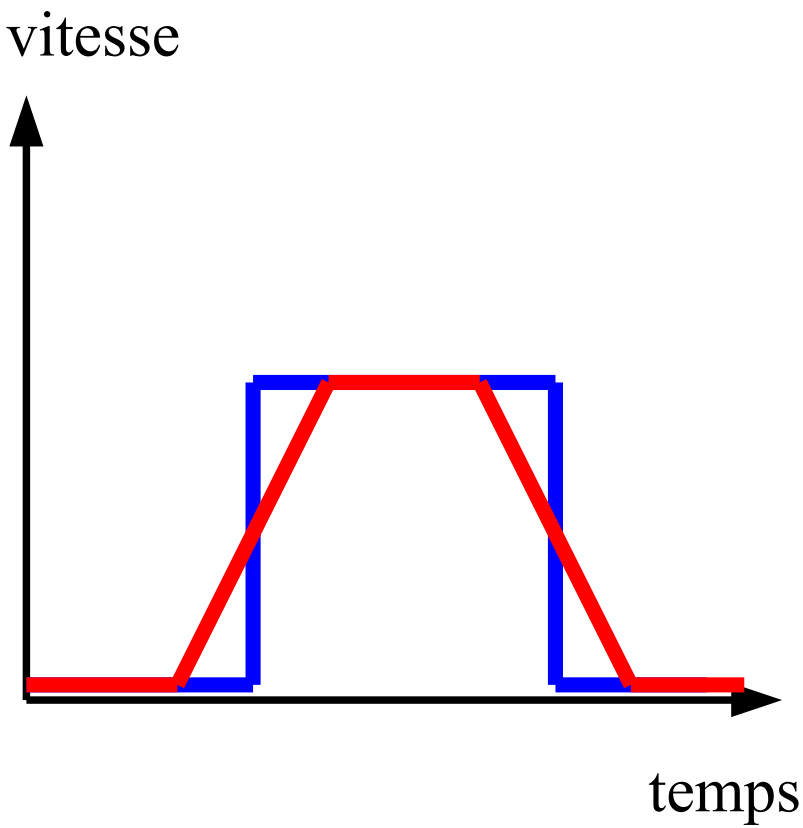
- Course



- Modèles du monde

Loi du monde (3)

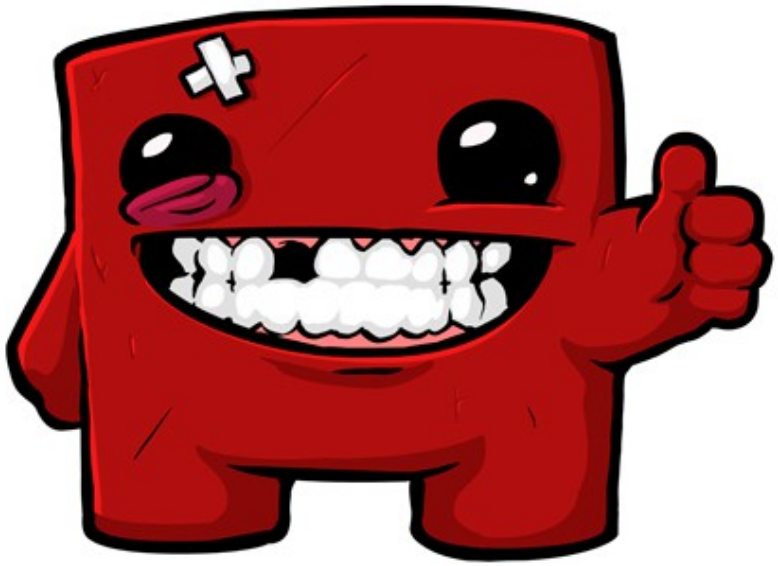
- Course



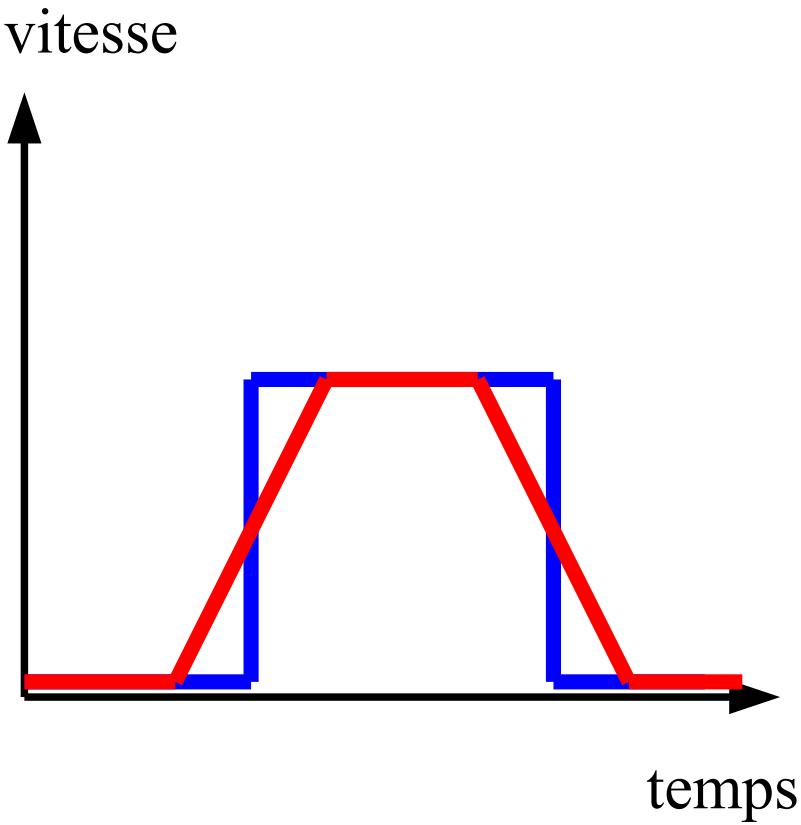
- Modèles du monde

Loi du monde (3)

- Course



Super meat boy - (indépendant)



- Modèles du monde

Modèles physiques

- Mécanique du point
  - Position, Vitesse, Accélération

- A chaque pas de temps

  a = donnée

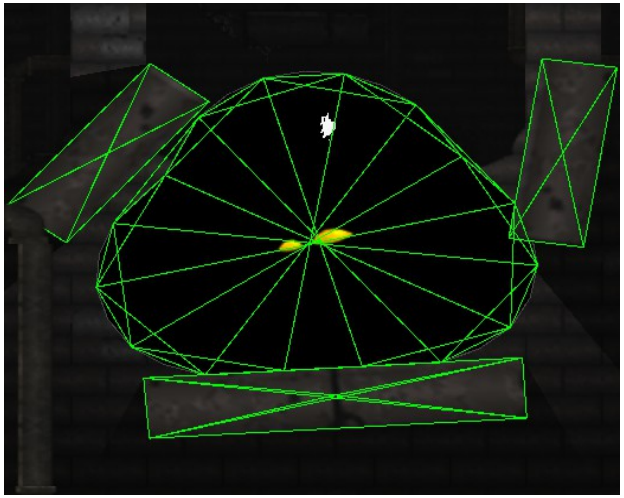
  v = v + a.dt

  x = x + v.dt

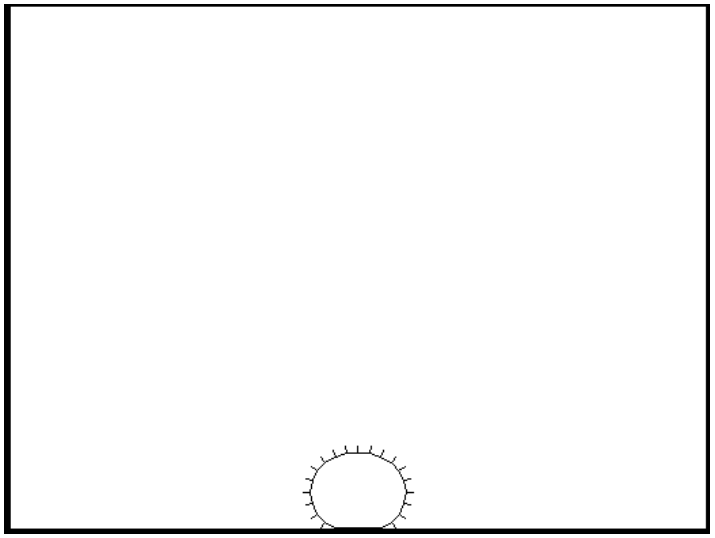
- Système masse-ressort

Modèles physiques

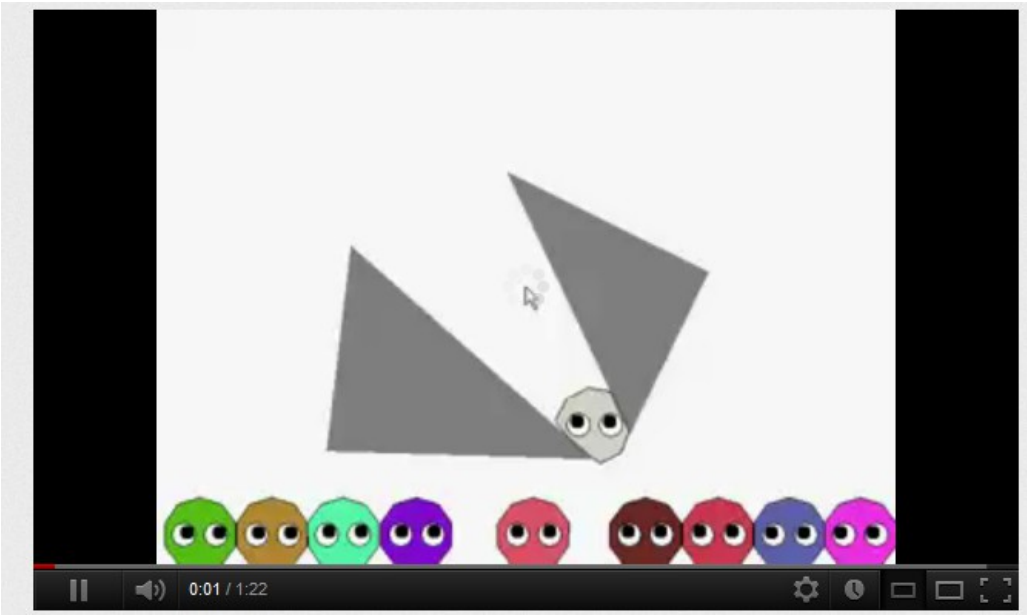
Gish



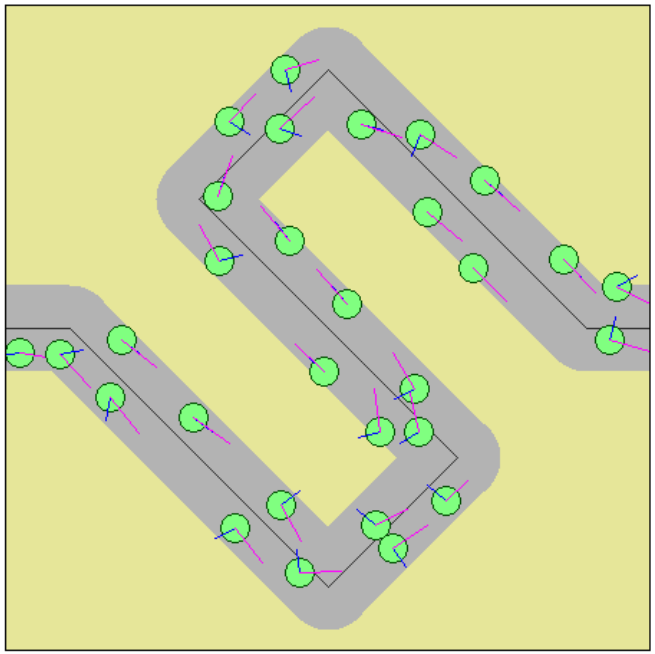
Modèles physiques

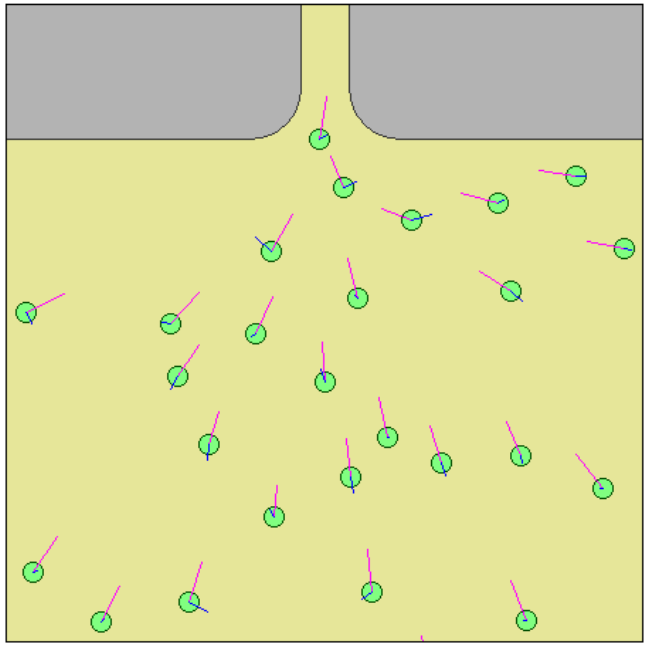


Kotsoft (youtube)



Modèles physiques





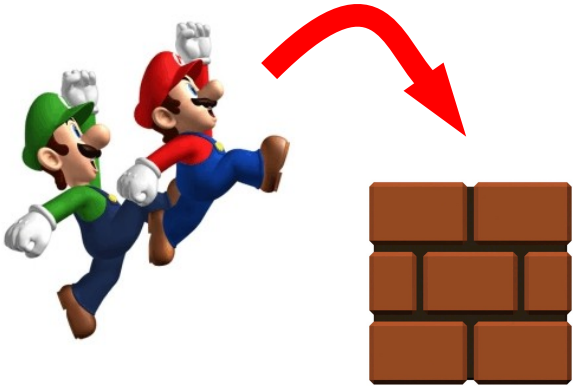
Steering behaviour
http://www.red3d.com/cwr/steer/

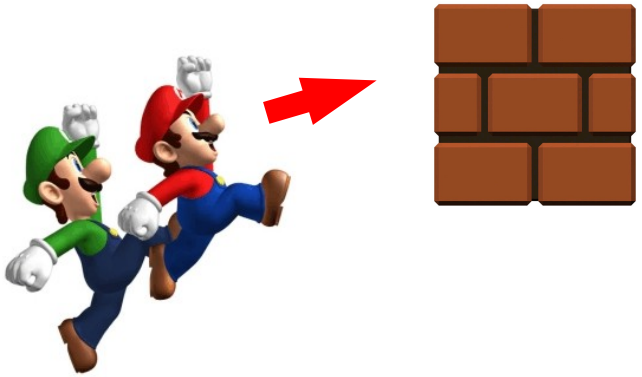
Démonstration Partie 3

Modèle physique

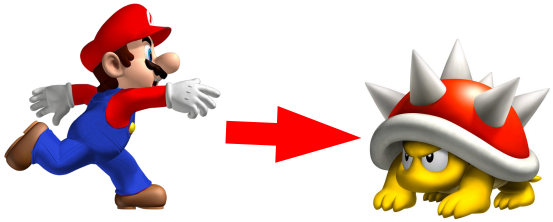
Gestion des collisions

- Interactions entre objets
  - Fixes





  - Dynamiques

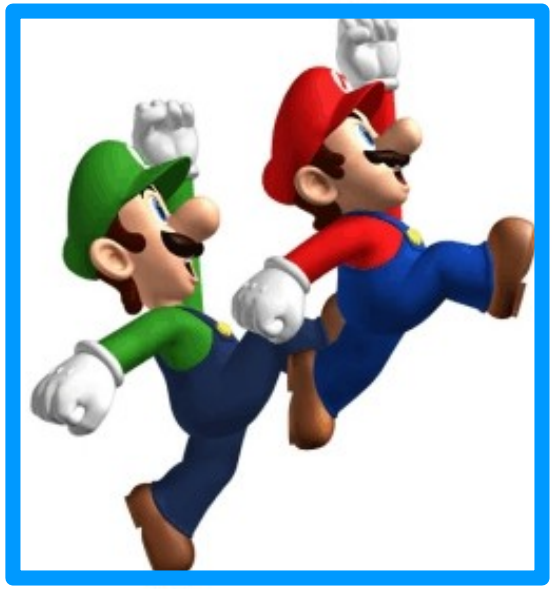


Gestionnaire de collision

- Principes
  - Utilisation de bounding box

Gestionnaire de collision

- Principes
  - Utilisation de bounding box

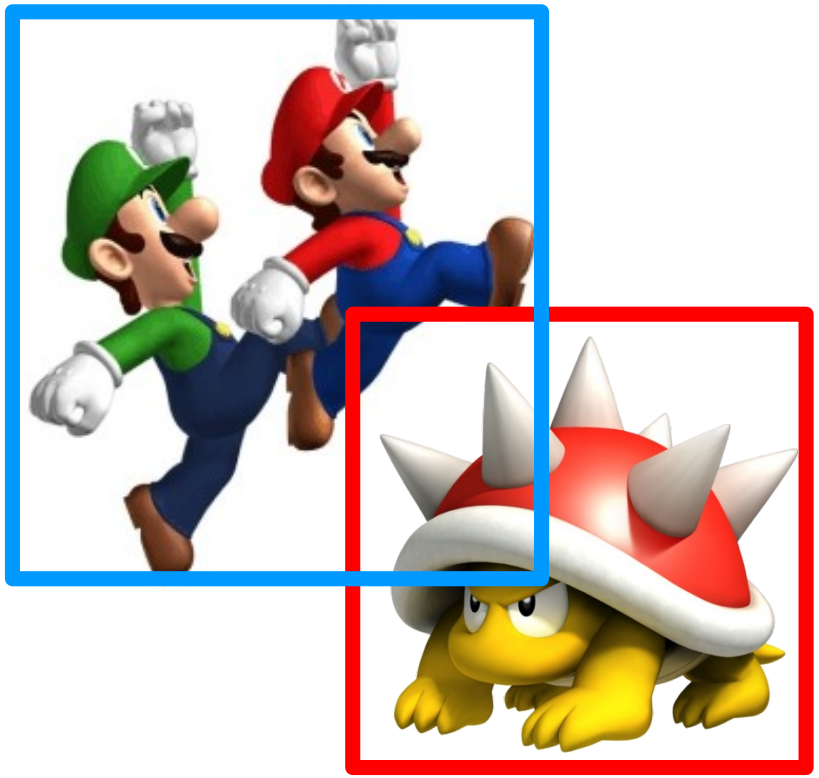


Gestionnaire de collision

- Principes
  - Utilisation de bounding box

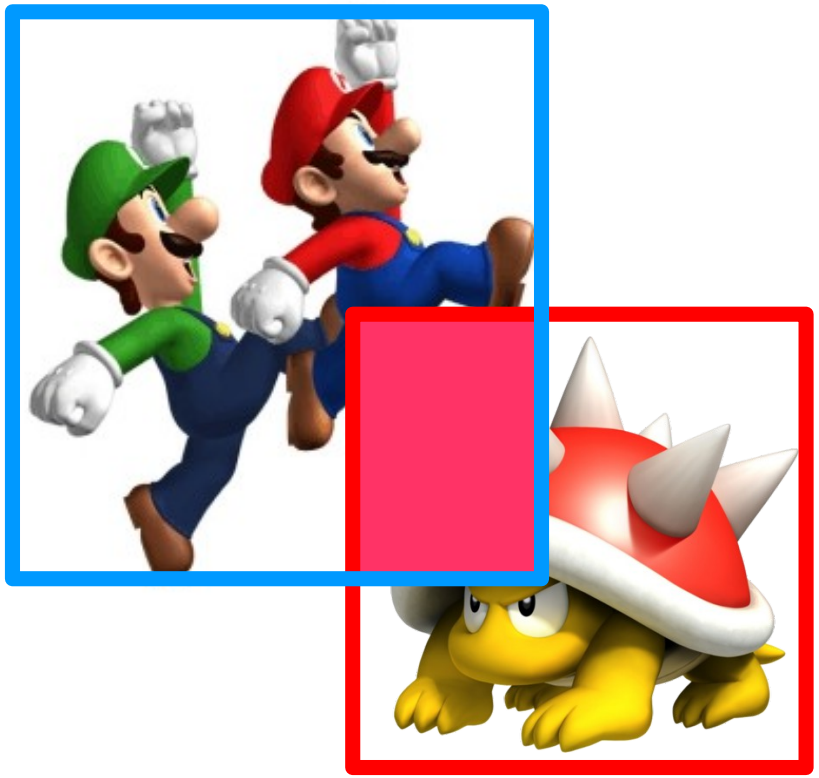
Gestionnaire de collision

- Principes
  - Utilisation de bounding box



Gestionnaire de collision

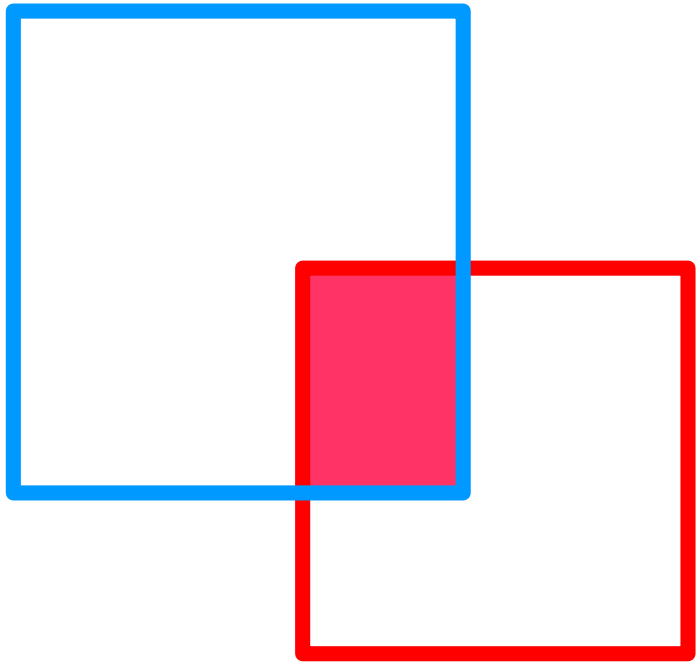
- Principes
  - Utilisation de bounding box



Gestionnaire de collision

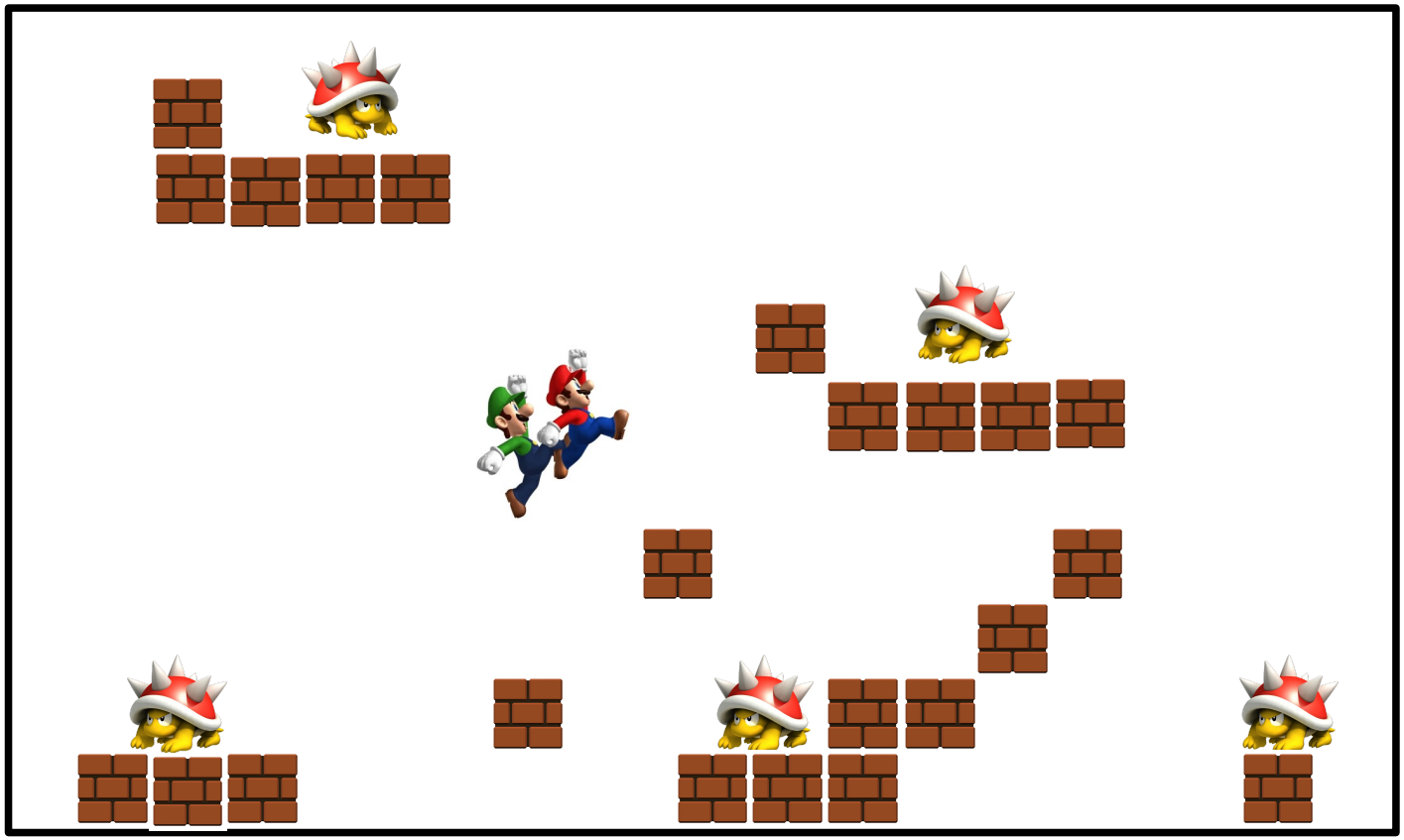
- Intersection de rectangles

```
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;
}
```



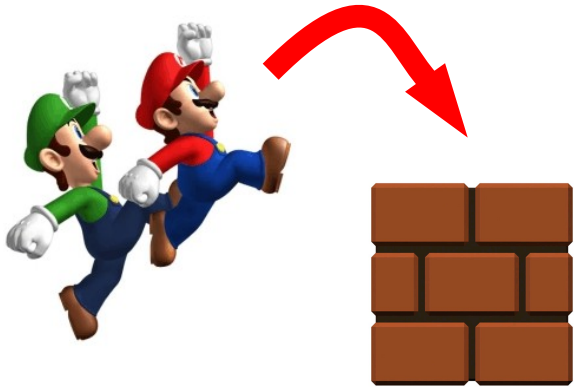
Gestionnaire de collision

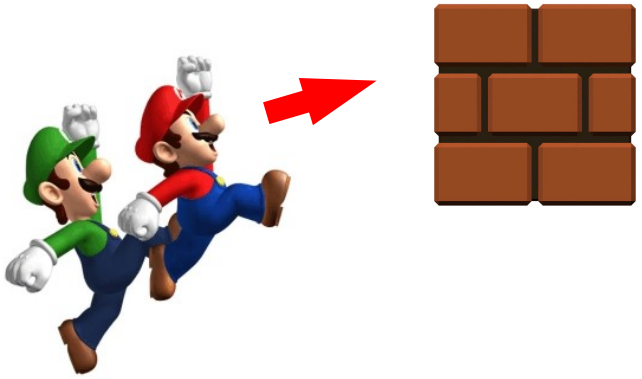
- Question ouverte
  - recherche des objets proches ?



Gestionnaire de collision

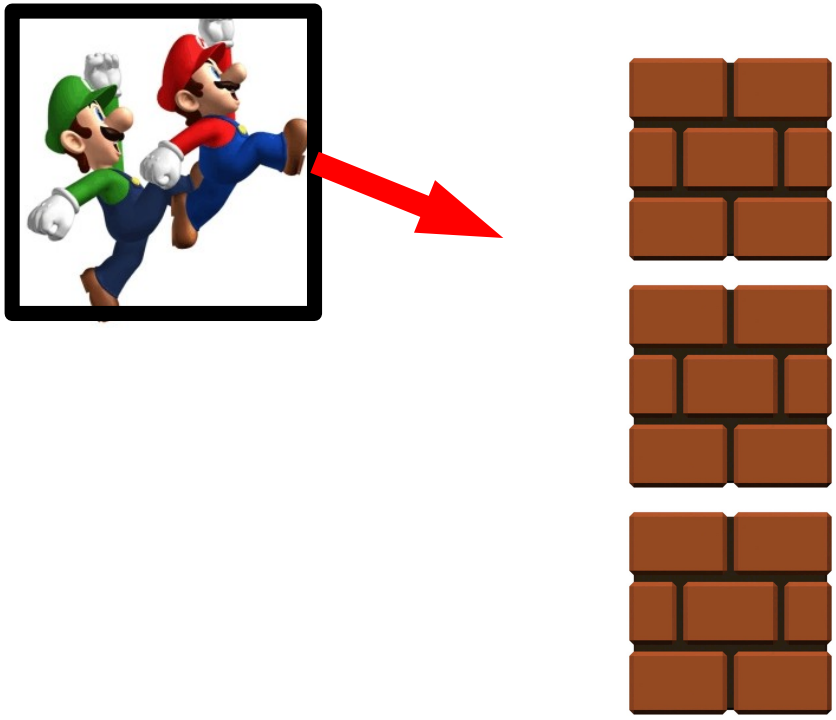
- Prise en compte du résultat
  - Dépend de l'angle





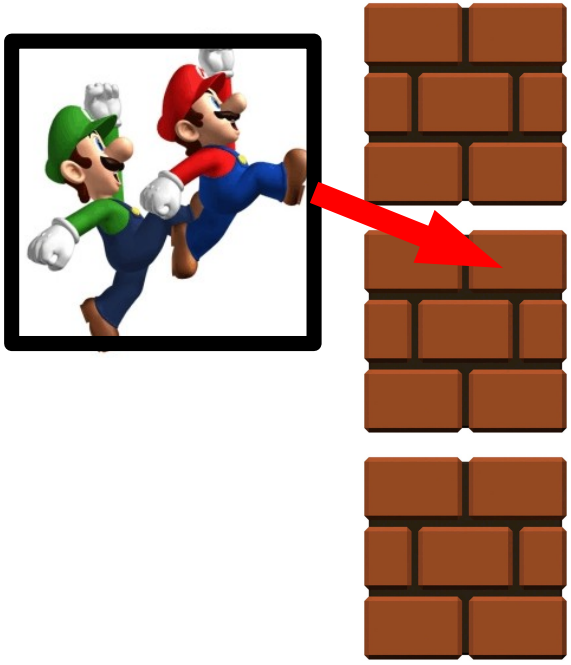
Gestionnaire de collision

- Prise en compte du résultat



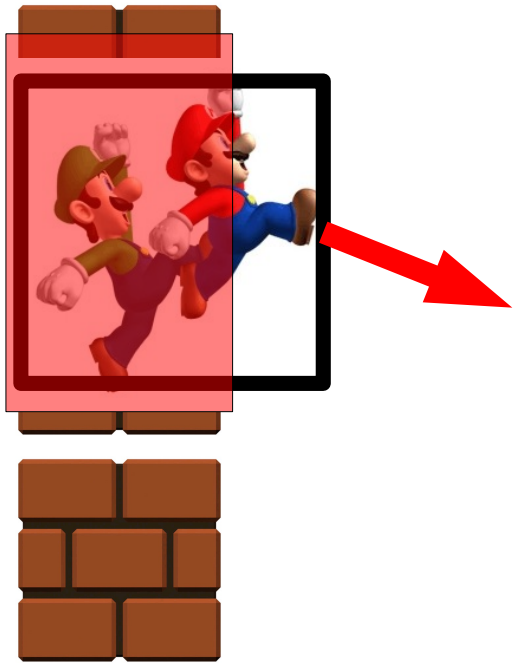
Gestionnaire de collision

- Prise en compte du résultat



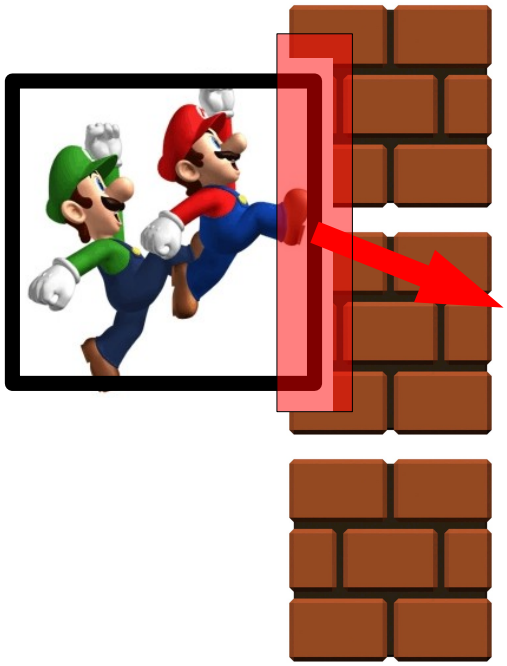
Gestionnaire de collision

- Prise en compte du résultat
  - Peu indétermination (t petit)



Gestionnaire de collision

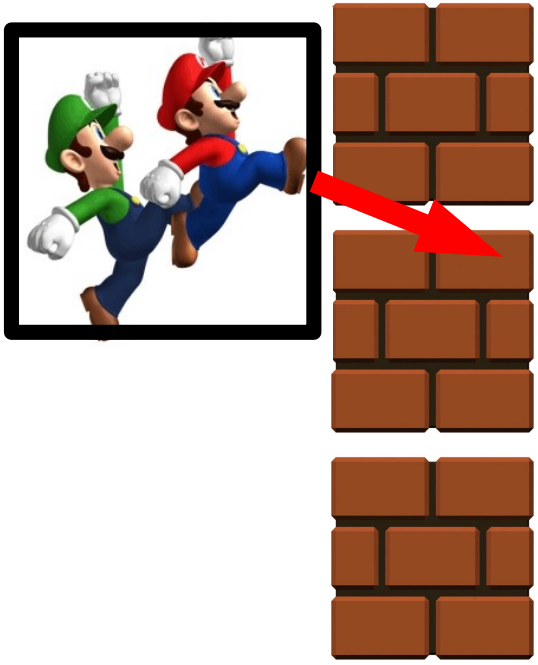
- Prise en compte du résultat
  - Peu indétermination (t petit)



Gestionnaire de collision

- Prise en compte du résultat
  - Peu indétermination (t petit)

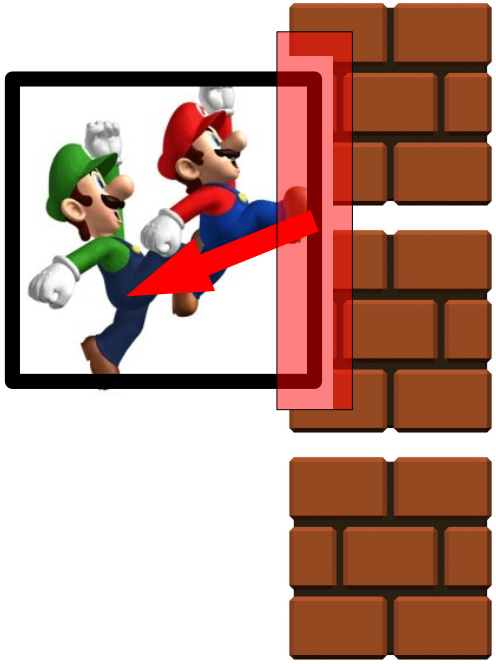
- Solution
  1. Revenir passé



Gestionnaire de collision

- Prise en compte du résultat
  - Peu indétermination (t petit)

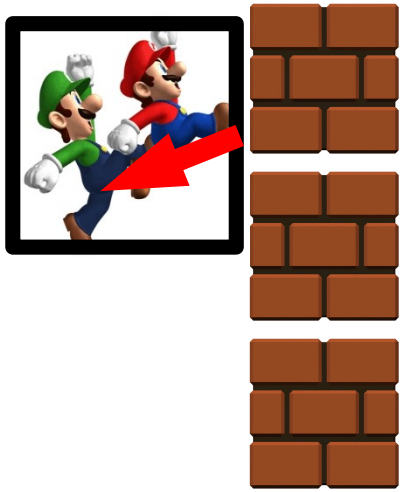
- Solution
  1. Revenir passé
  2. Approximation

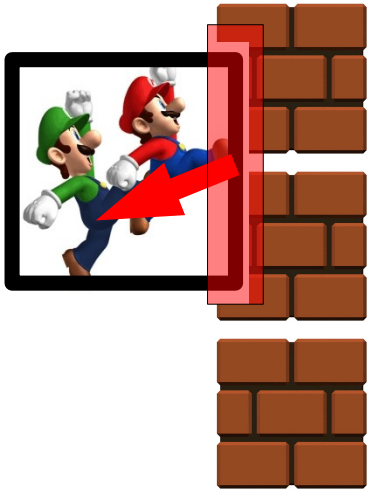


Gestionnaire de collision

- Prise en compte du résultat
  - Peu indétermination (t petit)

- Solution
  1. Revenir passé
  2. Approximation

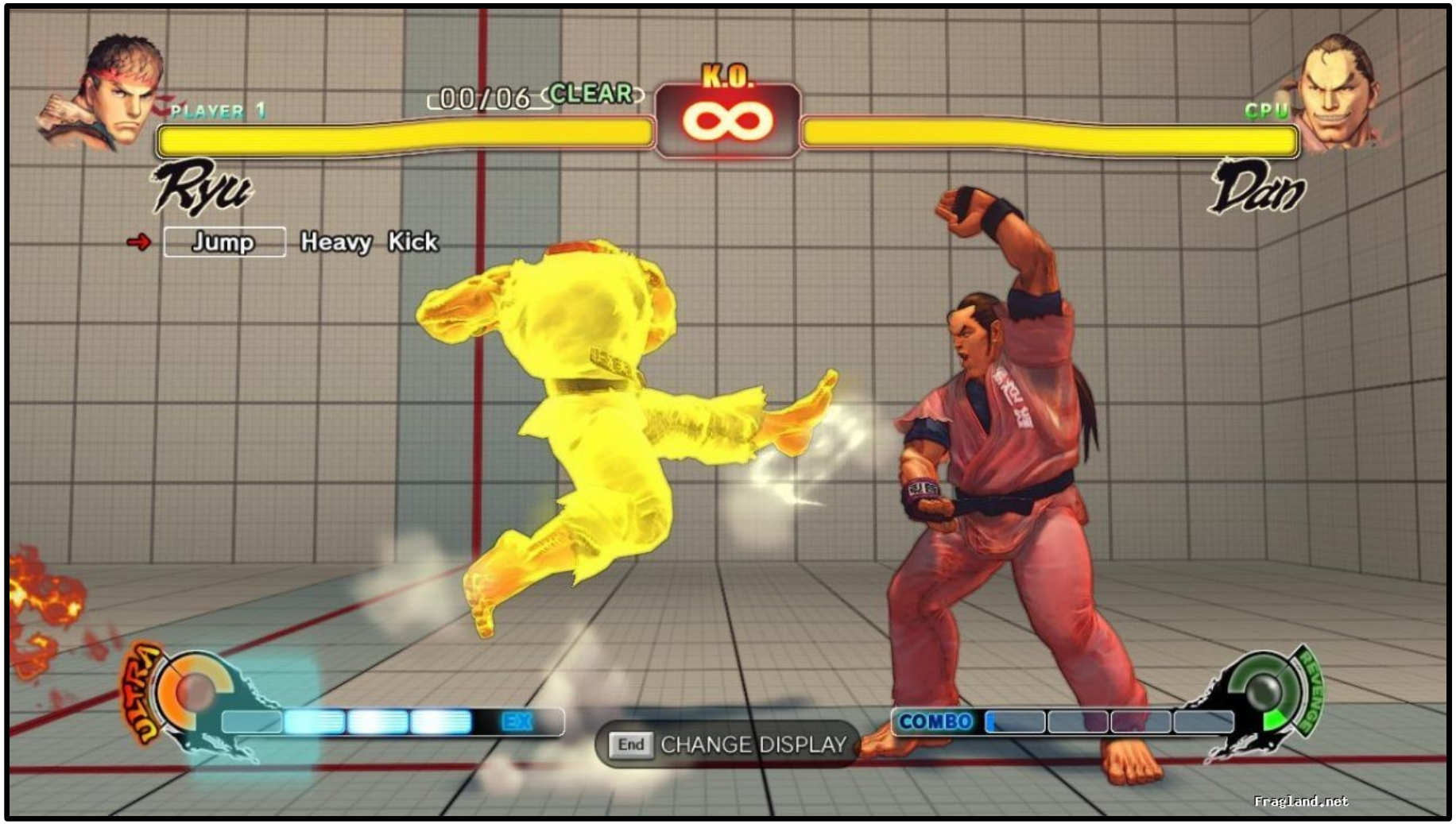




Bilan

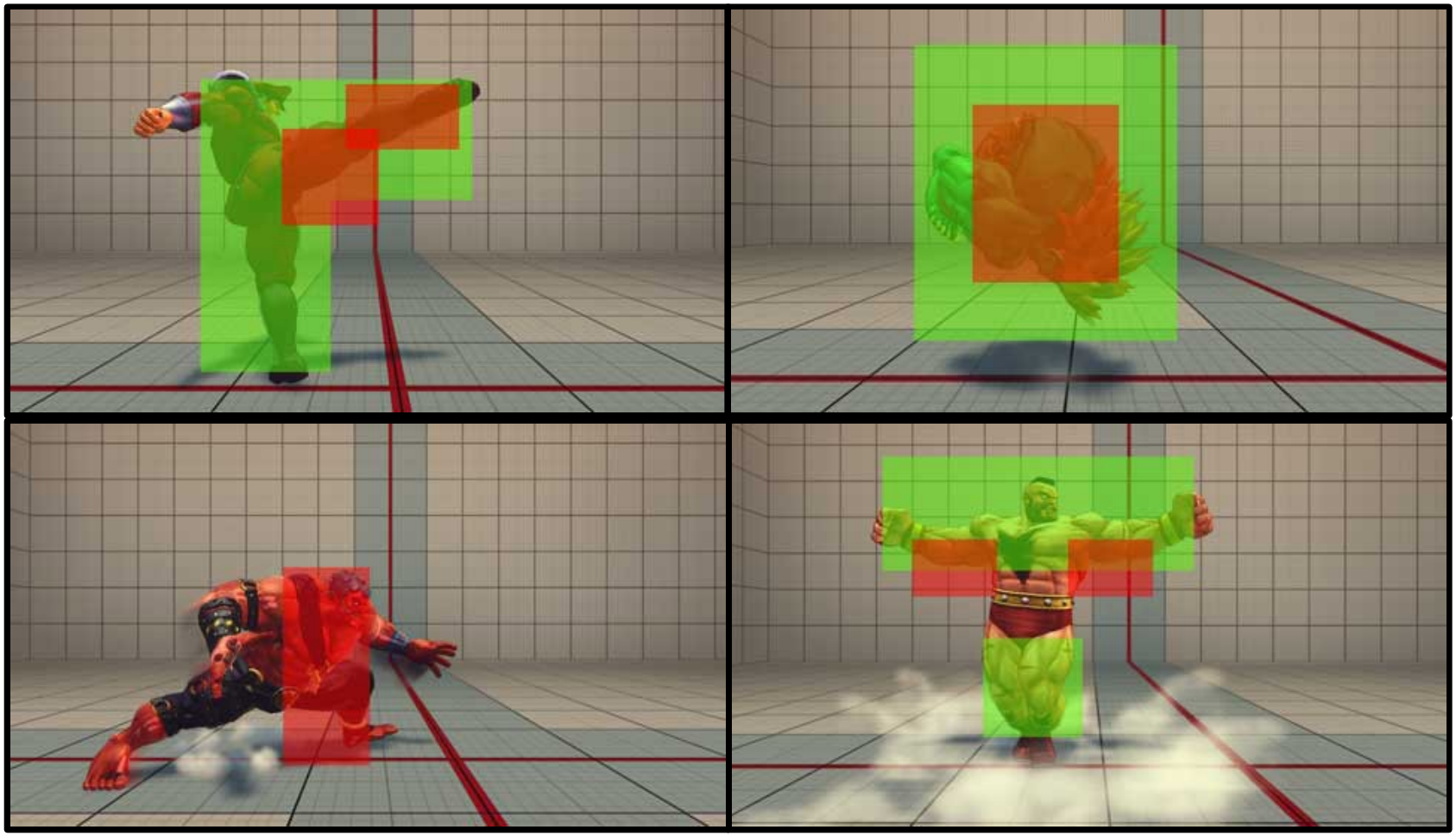
- Gestionnaire de collision
  - Élément de base dans jeu
  - Monde continu
- Génère les événements
  - Rencontre
  - Obstacles
- Présent dans la plupart des jeux

Jeux de combat 2D(.5)



Jeux de combat 2D(.5)

- Hitbox / Hurtbox (ex Street Fighter 4)

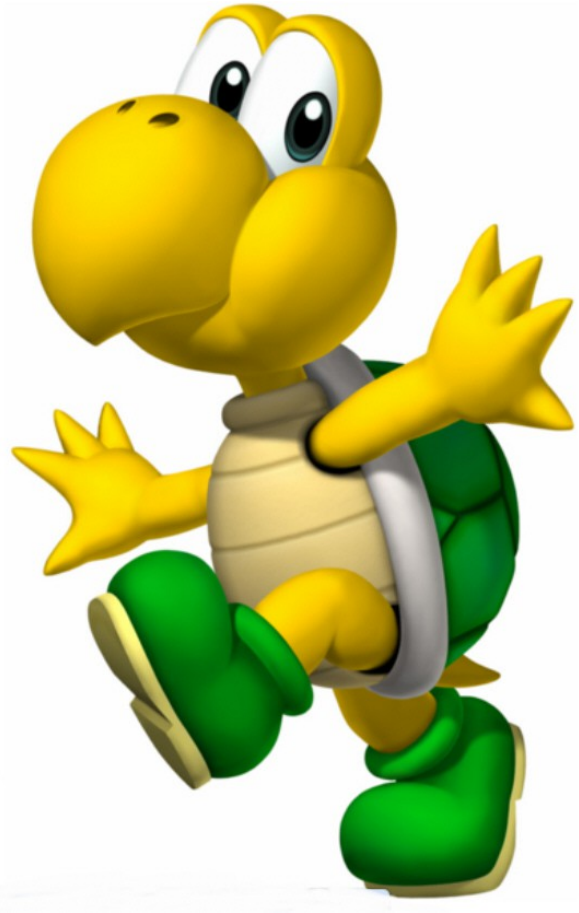


Démonstration Partie 4

Gestion des collisions

Intelligence Artificielle

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



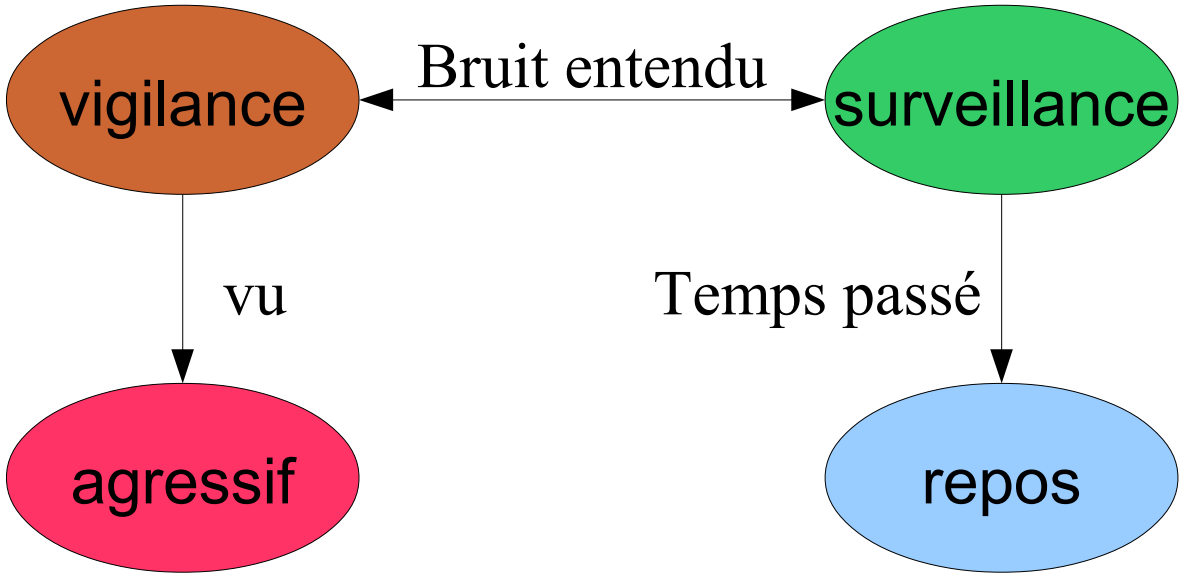
Intelligence Artificielle

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

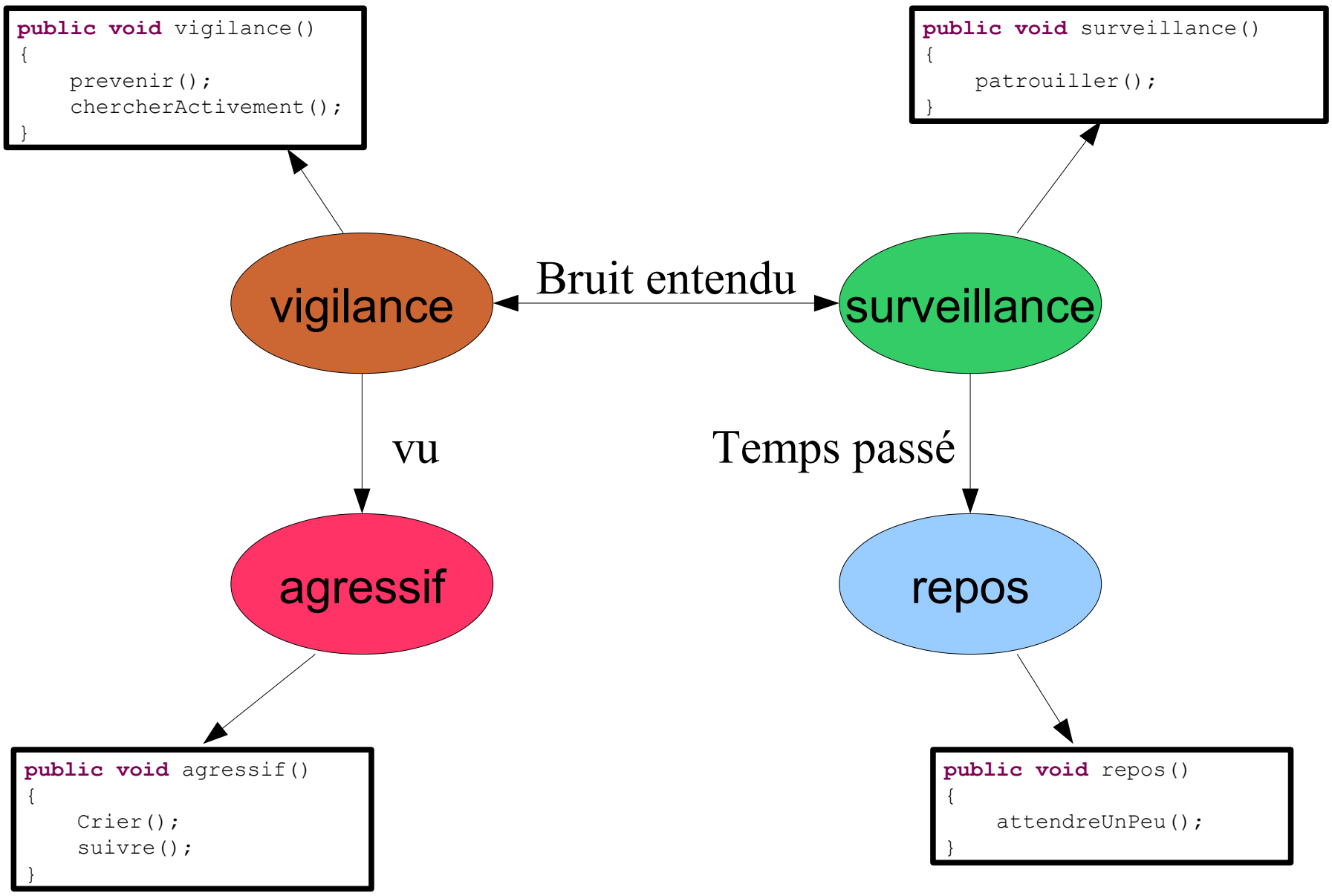

Beyond Good & Evil developer Ubi Soft publisher Ubi Soft
© GameWallpapers.com hosted by JTLnet.com

Exemple « beyond good and evil »

Exemple « beyond good and evil »



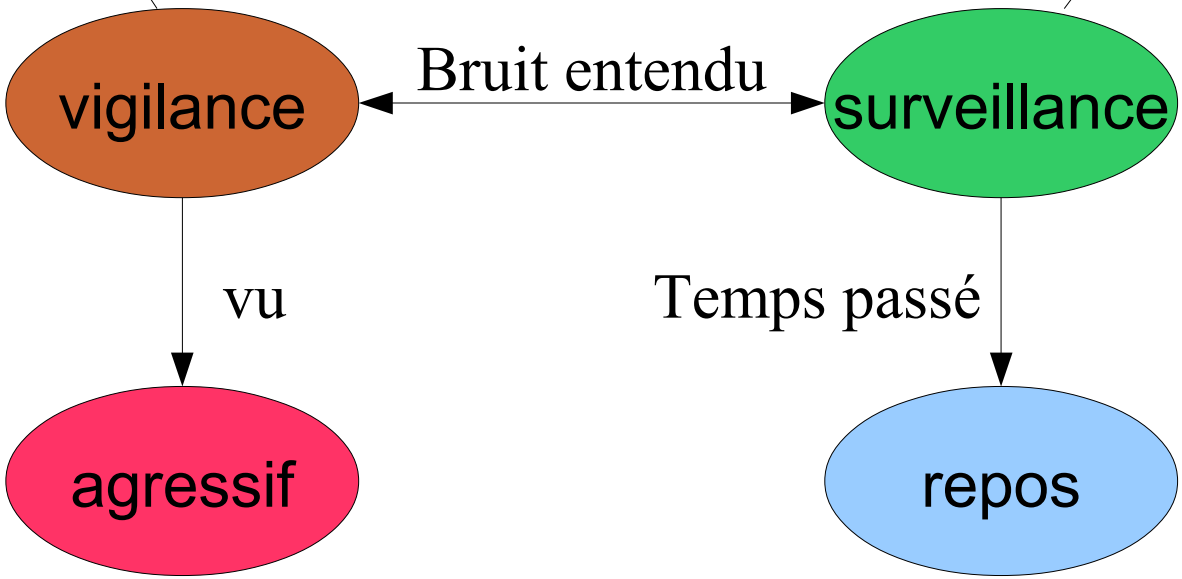
Exemple « beyond good and evil »



Exemple « beyond good and evil »


```
public void vigilance()
{
    prevenir();
    ChercherActivement();

    if (vu) cmpt=agressif
}
```

```
public void surveillance()
{
    Patrouiller();

    if (bruit) cmpt=vigilant
}
```



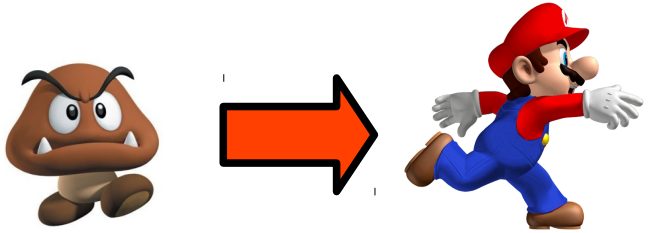

```
public void agressif()
{
    Crier();
    Suivre();

    if (tué) cmpt=surveille
}
```

```
public void repos()
{
    AttendreUnPeu();

    if (duree>100) cmpt=surveille
}
```

Un exemple

- Deux comportements
  - Promenade

  - Attaque



Un exemple

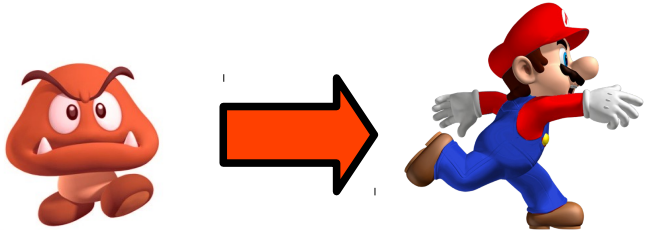
- Deux comportements
  - Promenade

"Promene"

Action
→ Avancer jusque obstacle

Transition
→ Mario proche => attaque

  - Attaque



"Attaque"

Action
→ Suivre Mario
→ Monter obstacles

Transition
→ Mario trop loin => promène

Code IA

- 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;
  }
}
```

Code IA

- 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);
}
```

Intelligence Artificielle

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

Démonstration Partie 5

Intelligence artificielle

Bilan de la partie modèle

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

Plan

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

Gestion des contrôleurs

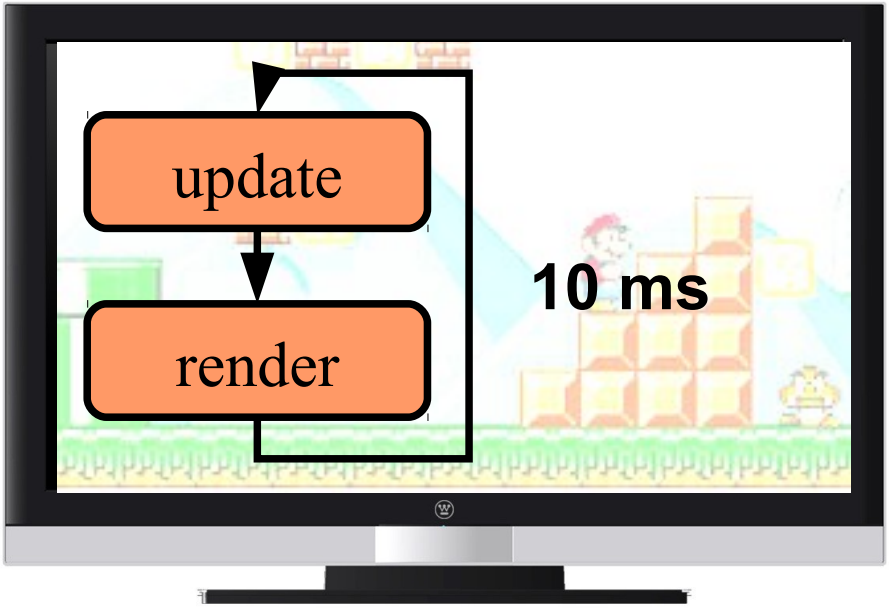
- Ne pas rater un événement
  - Mauvaise jouabilité
- Mise à jour rapide
  - Bloquant pour l'écoute
- Centraliser dans l'update
  - Pouvoir gérer facilement les interactions

Gestion des contrôles utilisateurs

- Actions du joueur

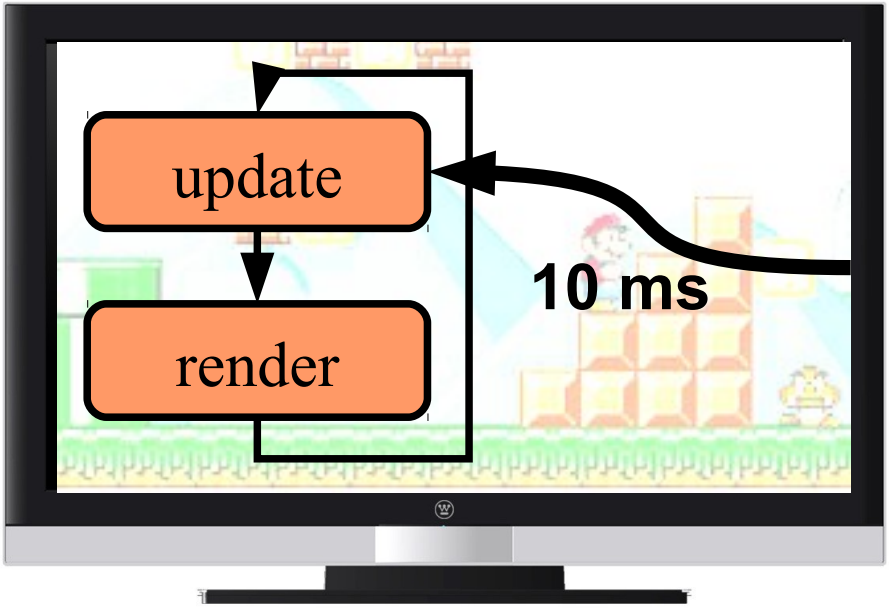
Gestion des contrôles utilisateurs

- Actions du joueur



Gestion des contrôles utilisateurs

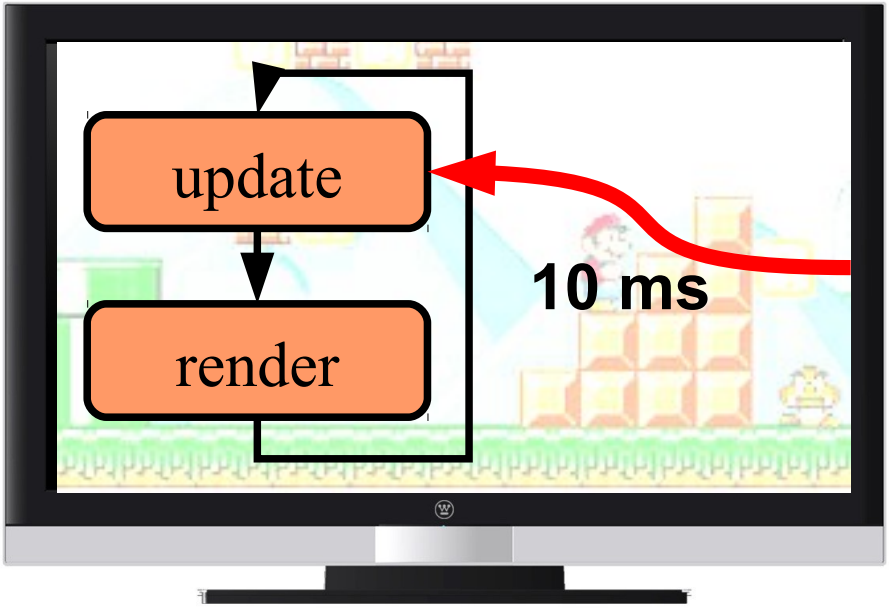
- Actions du joueur



Gestion des contrôles utilisateurs

- Actions du joueur

Or action est asynchrone



Principe

- 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-- ;
  }
}
```

Principe

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

Principe

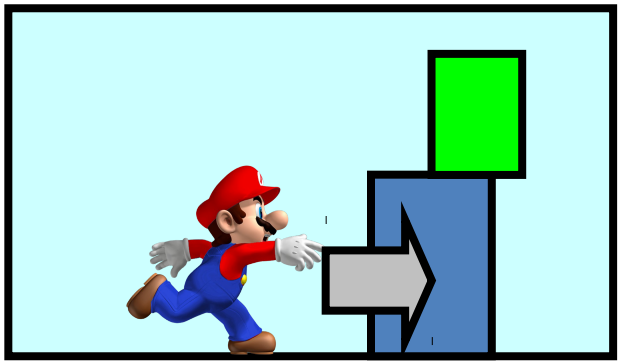
```
public void keyPressed(KeyEvent e) {
  if (e.getKeyCode() == e.VK_LEFT) {
    mario.x-- ;
  }
  if (e.getKeyCode() == e.VK_RIGHT) {
    mario.x-- ;
  }
}
```

Principe

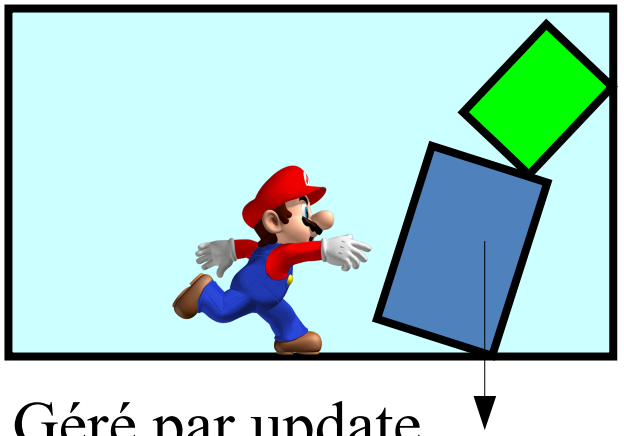
Problème concurrence d'accès

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

Intégrer le contrôle dans update



Géré par clavier



Géré par update

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

Problème du contrôle

- 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

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

```
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
     }
}
```

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;
}
```

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

Toujours de la concurrence mais ...

Intérêts

- 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

KeyListener

```
Gauche = true ;
```

Principe

MouseListener

```
Gauche = true ;
```

Principe

MouseListener

```
Gauche = true ;
```

Modalité d'interaction

(souris, clavier)

découplage

Règles du jeu

(conséquence action)

Différents types de contrôle

- 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

http://fivedots.coe.psu.ac.th/~ad/jg2/

Principe

Pad

```
Gauche = true ;
```

Principe

keyconfig

```
Gauche = true ;
```

Principe

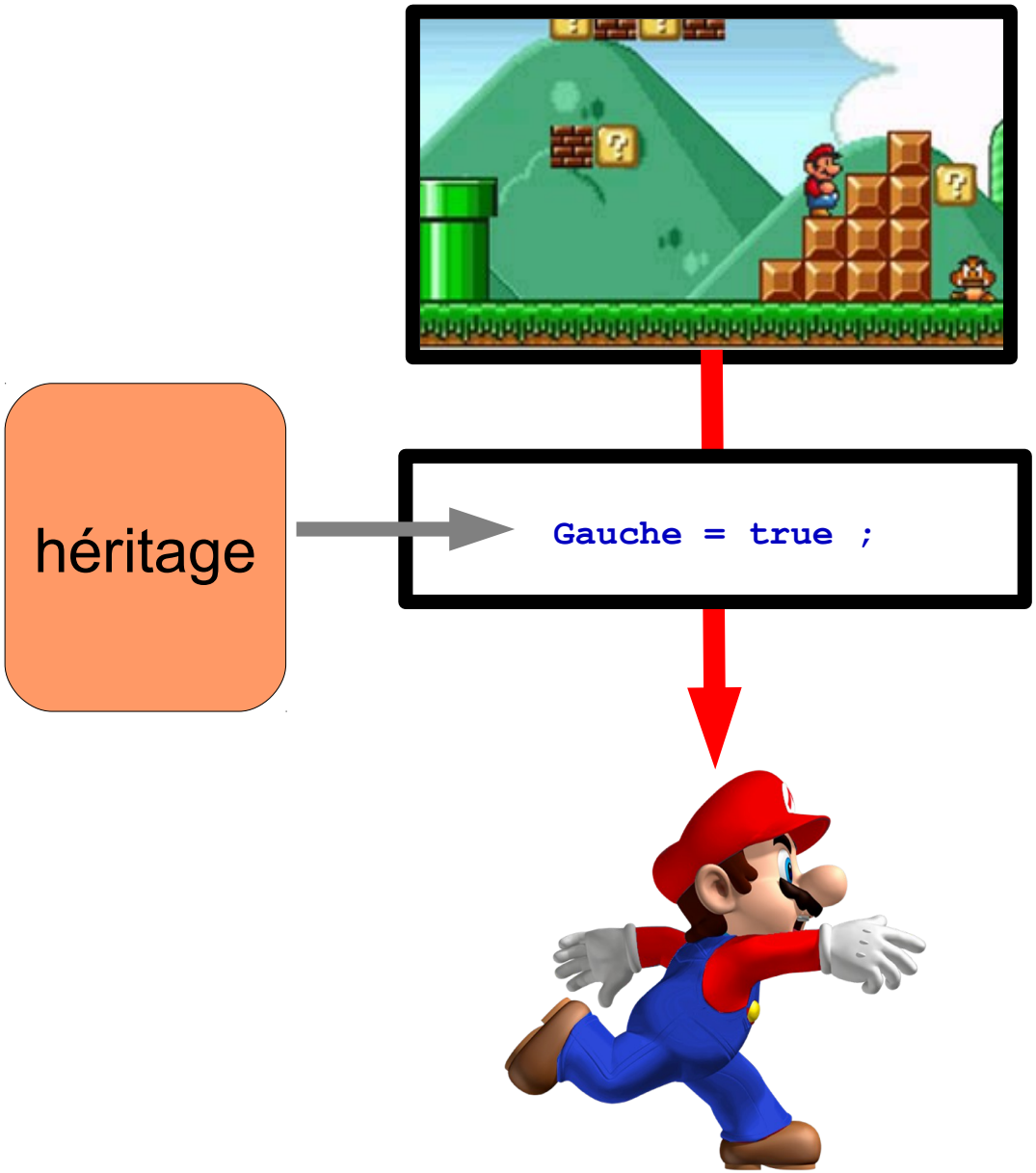
```
Gauche = true ;
```

Principe

héritage

Gauche = true ;

Principe



Principe

héritage

```
Gauche = true ;
```

IA Capable de remplacer un joueur

Démonstration Partie 6

Mise en œuvre d'un contrôleur Clavier / Souris / IA

Plan

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

Affichage et animation

- Techniques
  - Active rendereing
  - Double buffering
- Gestion de sprites
- Scrolling

Problèmes de l'affichage

- Intégrer le coût de l'affichage dans la boucle
  - Active rendering
- Faire un repaint rapide
  - Double buffering

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

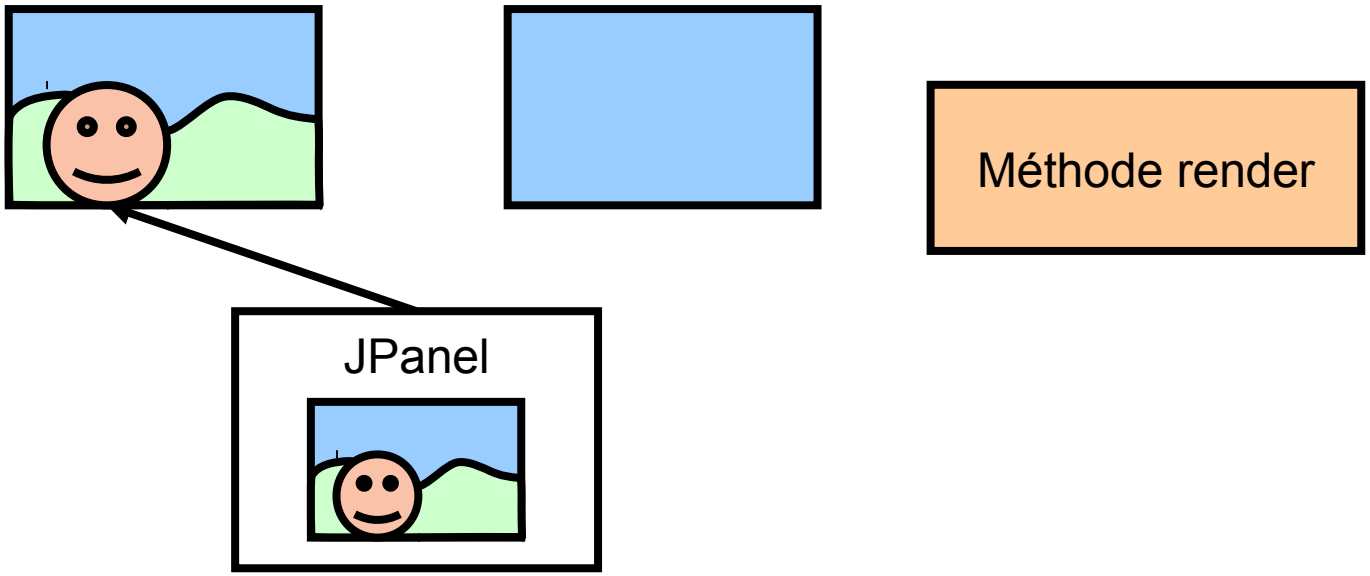
Active rendering

- Avec repaint()
  - Ne sait pas quand fini
  - Transformer repaint en une méthode

- Boucle principale
  - Méthode render
    - Récupère le graphics de l'image
    - Dessine dans le JPanel

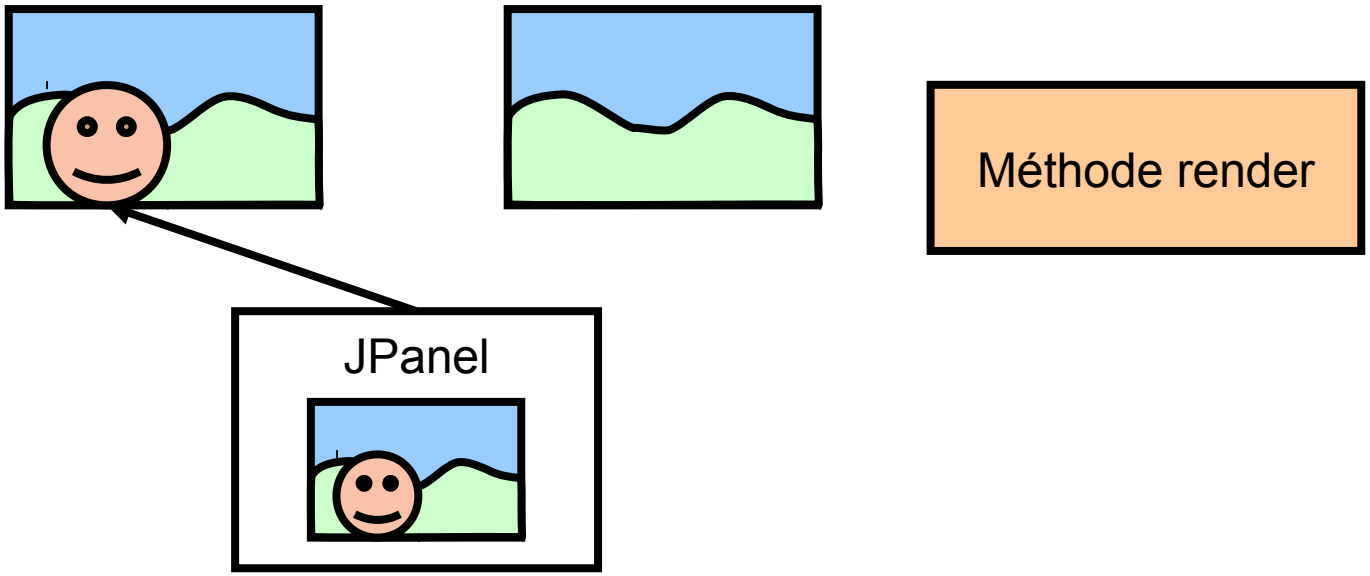
Double buffering

- Vitesse du paint importante
  - Modifie des images affichées ==> surbrillance
- Accélérer le rendu
  - Utiliser copie image



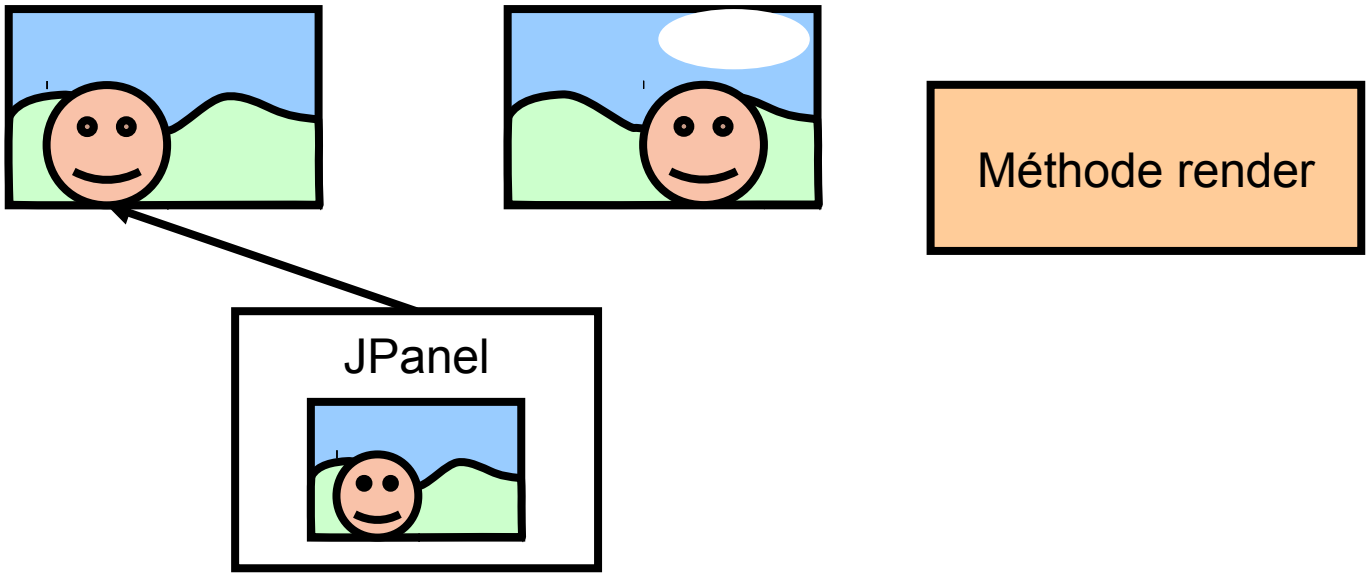
Double Buffering

- Vitesse du paint importante
  - Modifie des images affichées ==> surbrillance
- Accélérer le rendu
  - Utiliser copie image



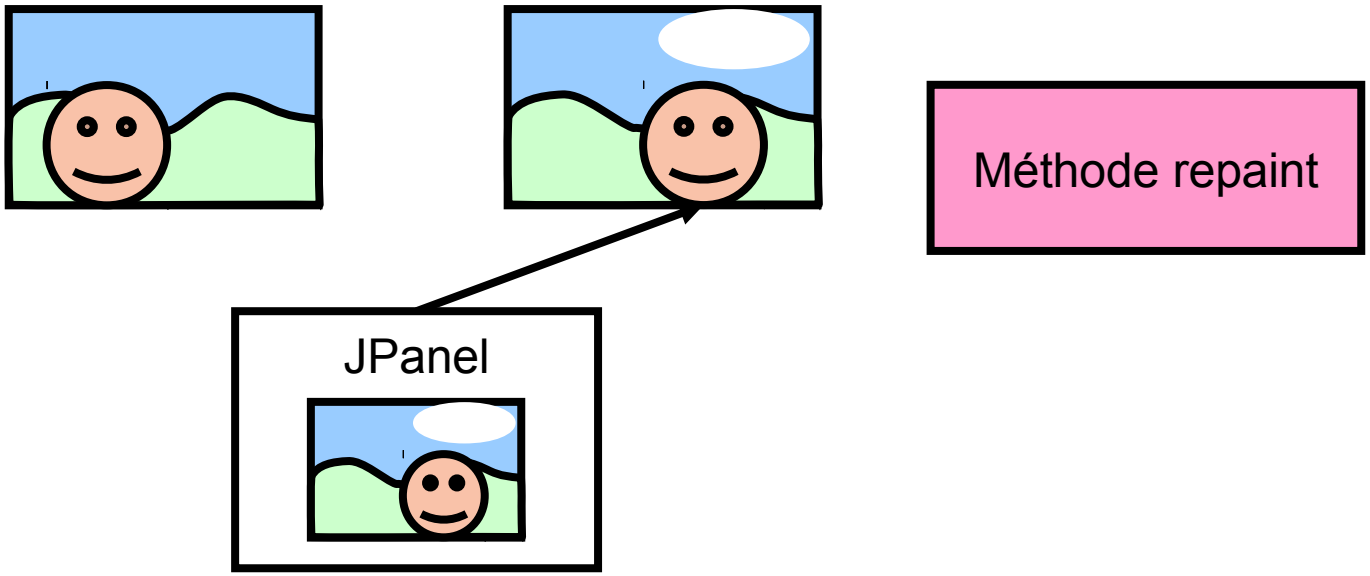
Double buffering

- Vitesse du paint importante
  - Modifie des images affichées ==> surbrillance
- Accélérer le rendu
  - Utiliser copie image



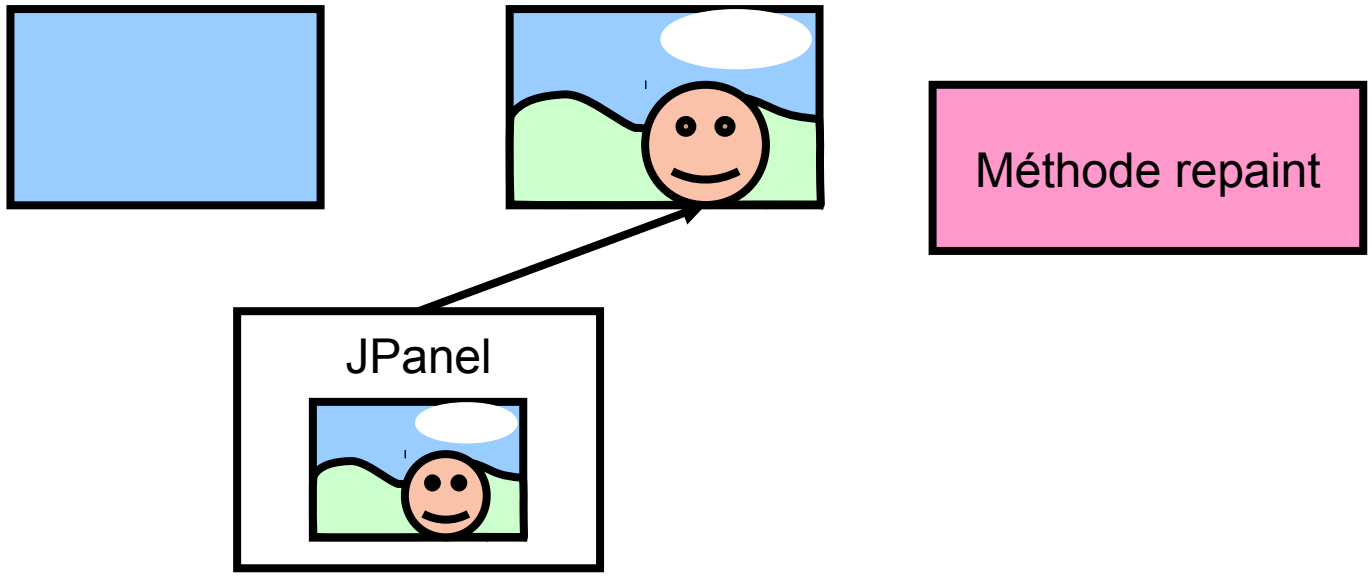
Double buffering

- Vitesse du paint importante
  - Modifie des images affichées ==> surbrillance
- Accélérer le rendu
  - Utiliser copie image



Double buffering

- Vitesse du paint importante
  - Modifie des images affichées ==> surbrillance
- Accélérer le rendu
  - Utiliser copie image



Double buffering

- Java
  - Class Bufferstrategy
- À partir Jframe
  - Synchronisation verticale
  - Plusieurs buffers

Démonstration partie 7

Double buffering

Animation - sprites

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

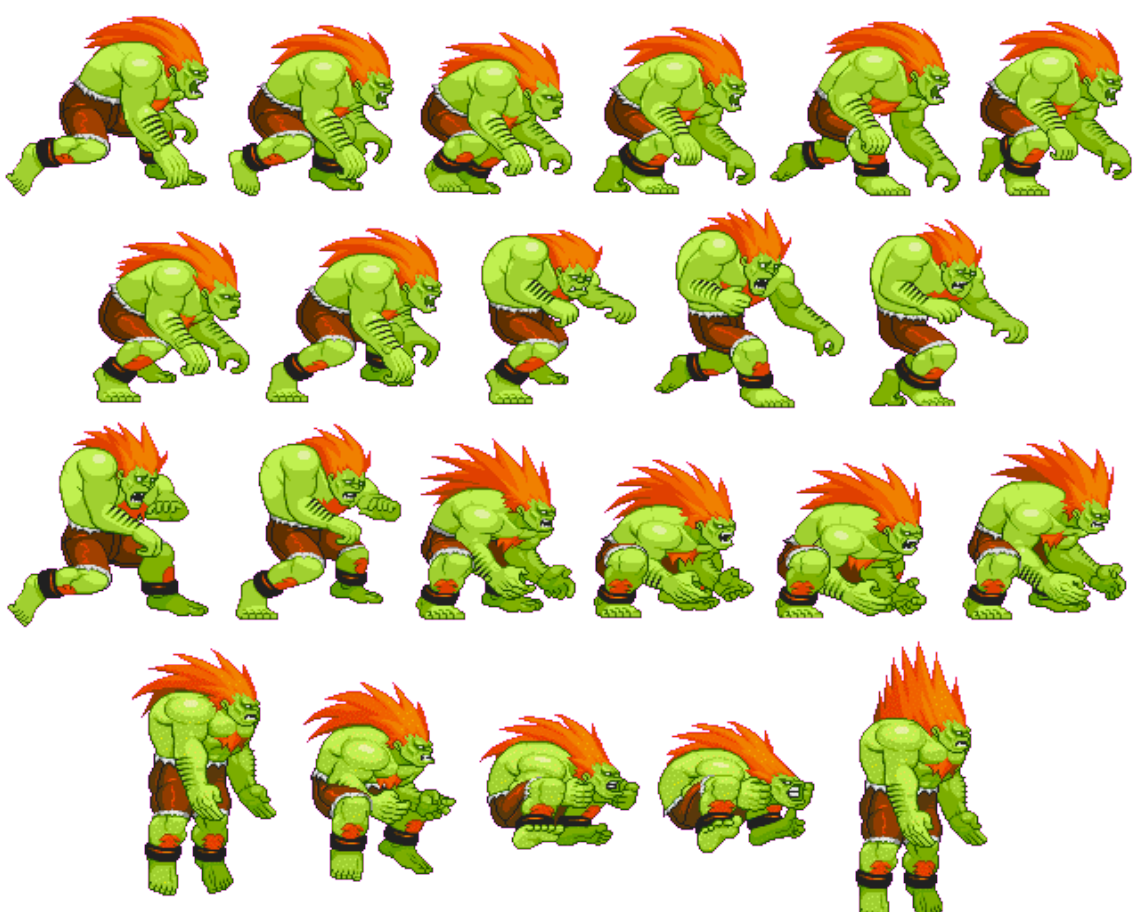
Animation - sprites

- 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 sprites

- Utilisation de Sprite
  - Extraction
  - Copie au bon endroit

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

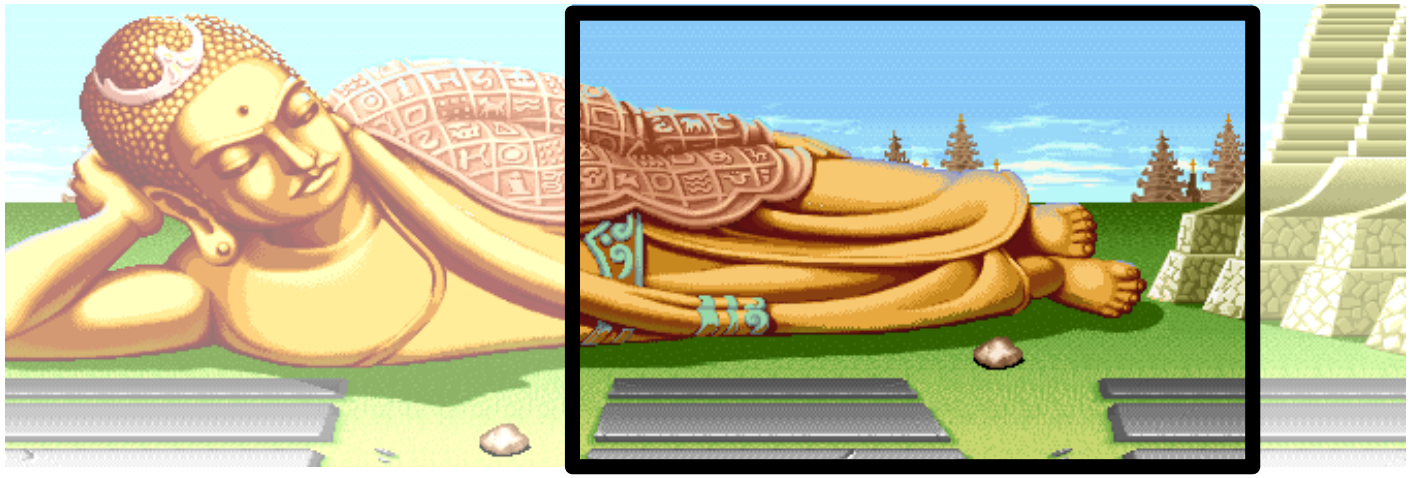
Utilisation de sprites

- Utilisation de Sprite
  - Extraction
  - Copie au bon endroit

Décor (Street fighter 2)

Utilisation de sprites

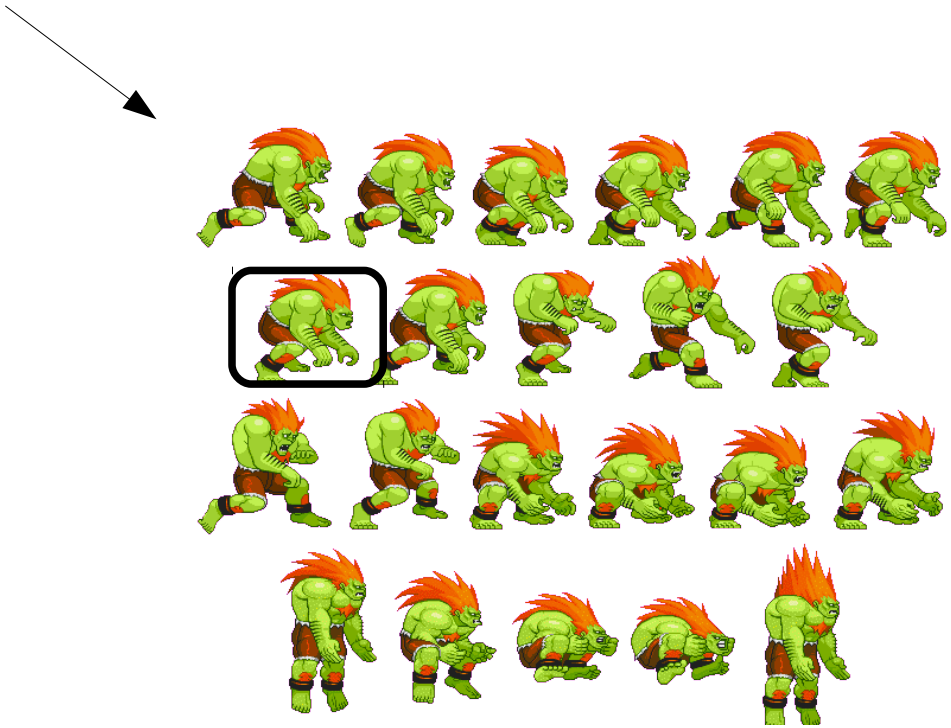
- Utilisation de Sprite
  - Extraction
  - Copie au bon endroit



Extraction du décor à afficher

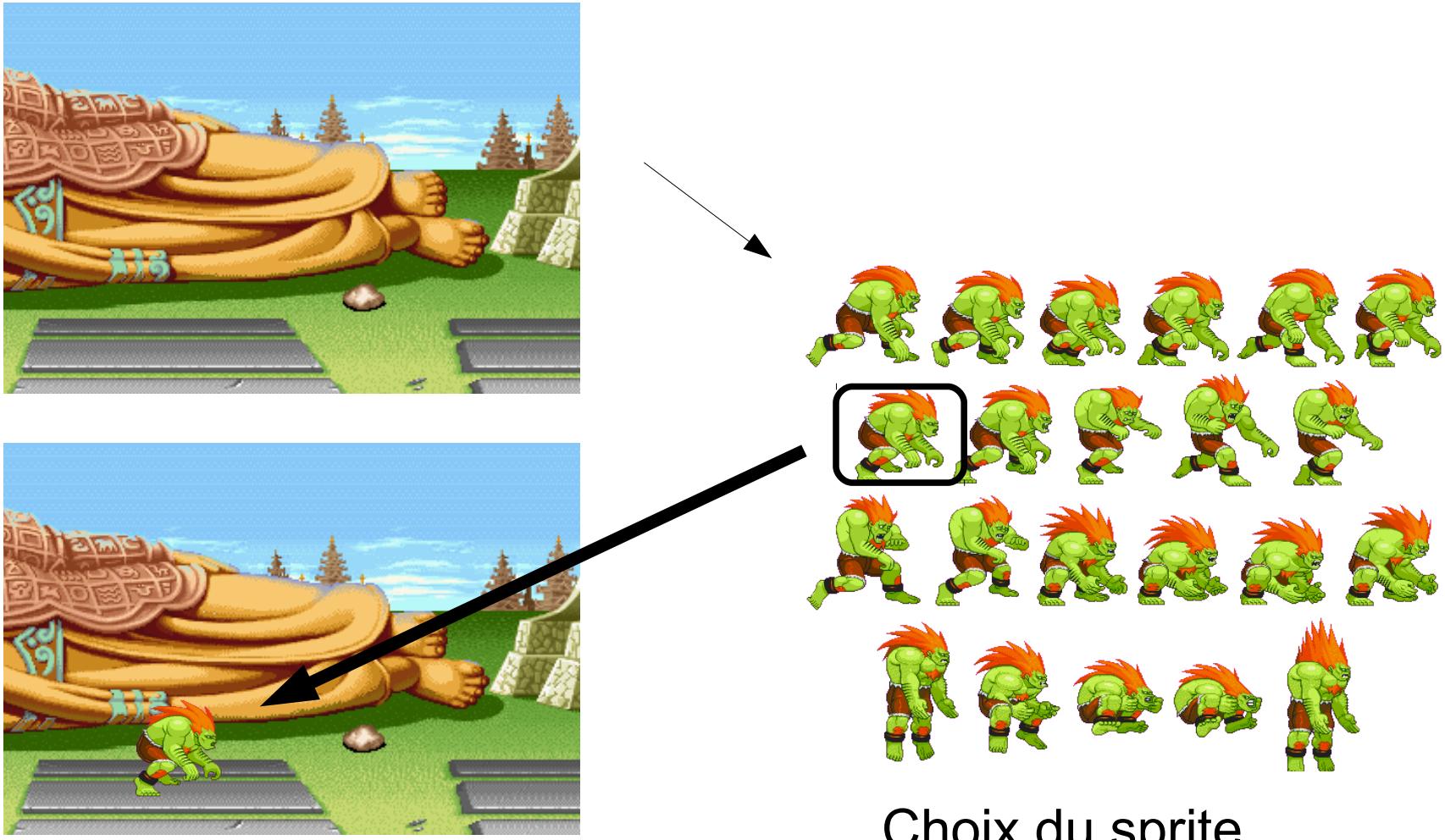
Utilisation de sprites

- Choix du sprite



Utilisation de sprites

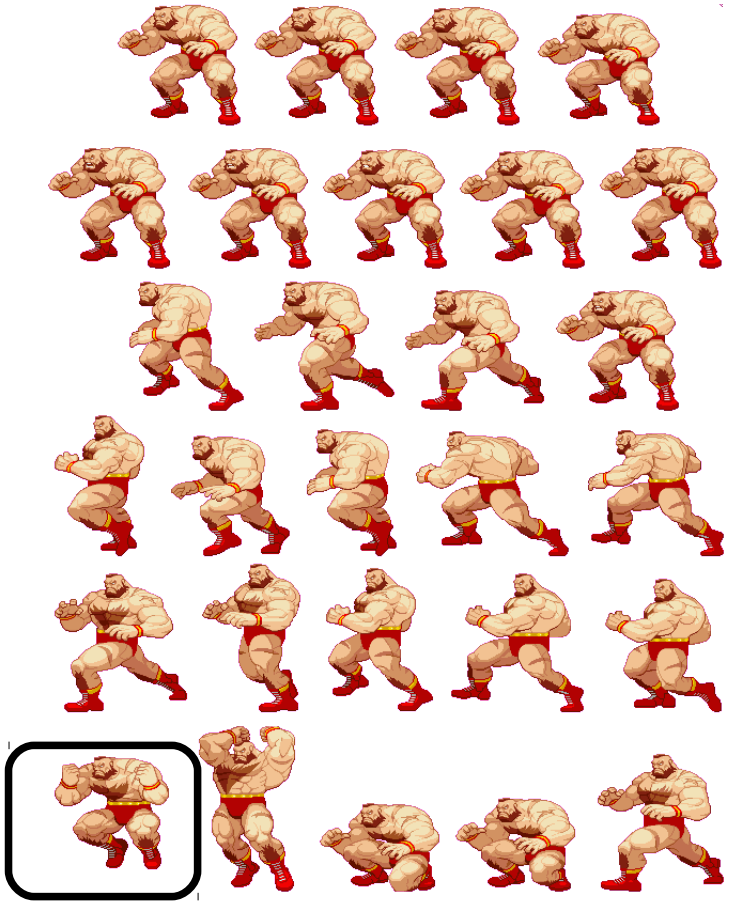
- Copie + gestion transparence



Choix du sprite

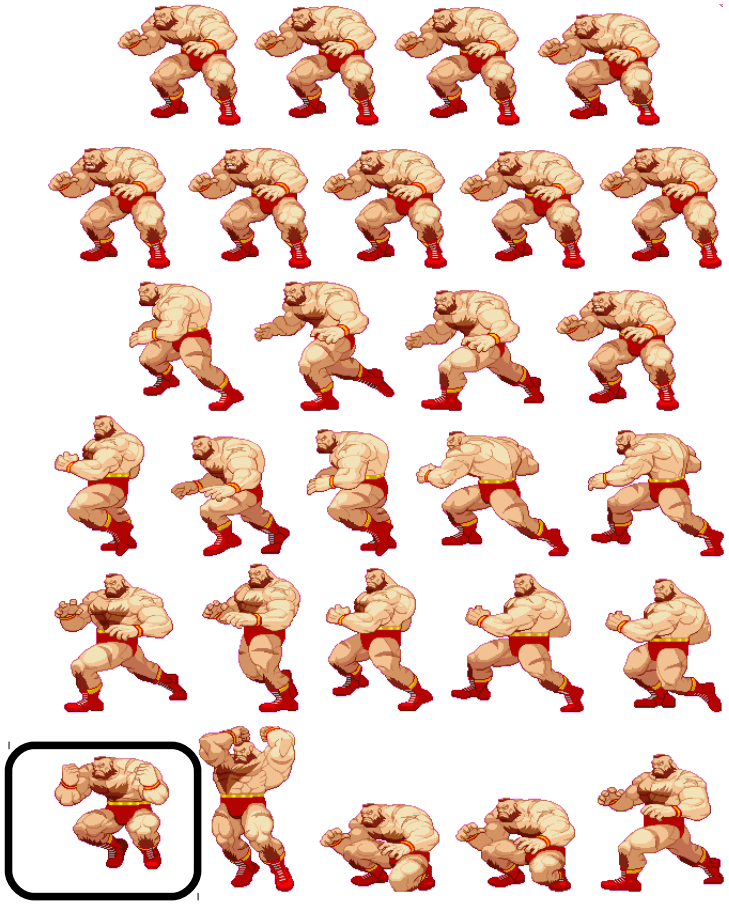
Utilisation de sprites

- Choix sprite (bis)



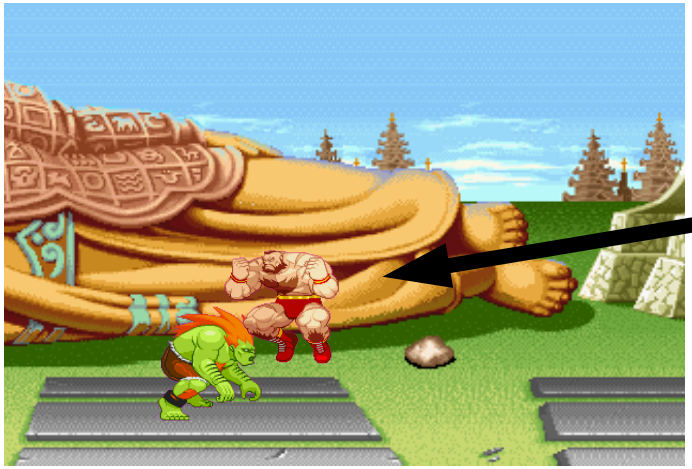
Utilisation de sprites

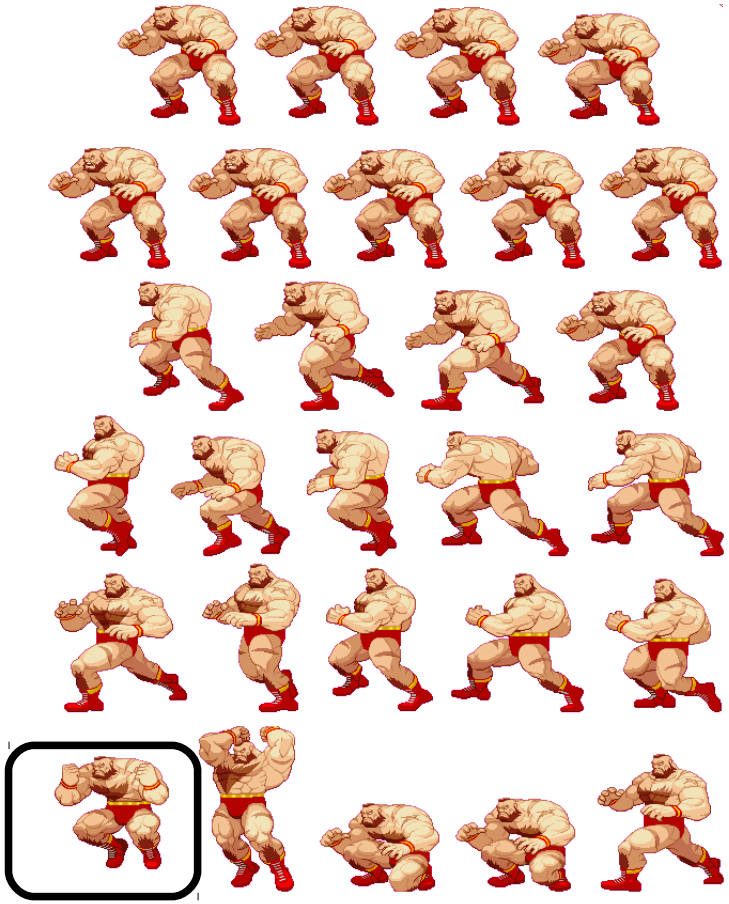
- Choix sprite (bis)



Utilisation de sprites

- Choix sprite (bis)



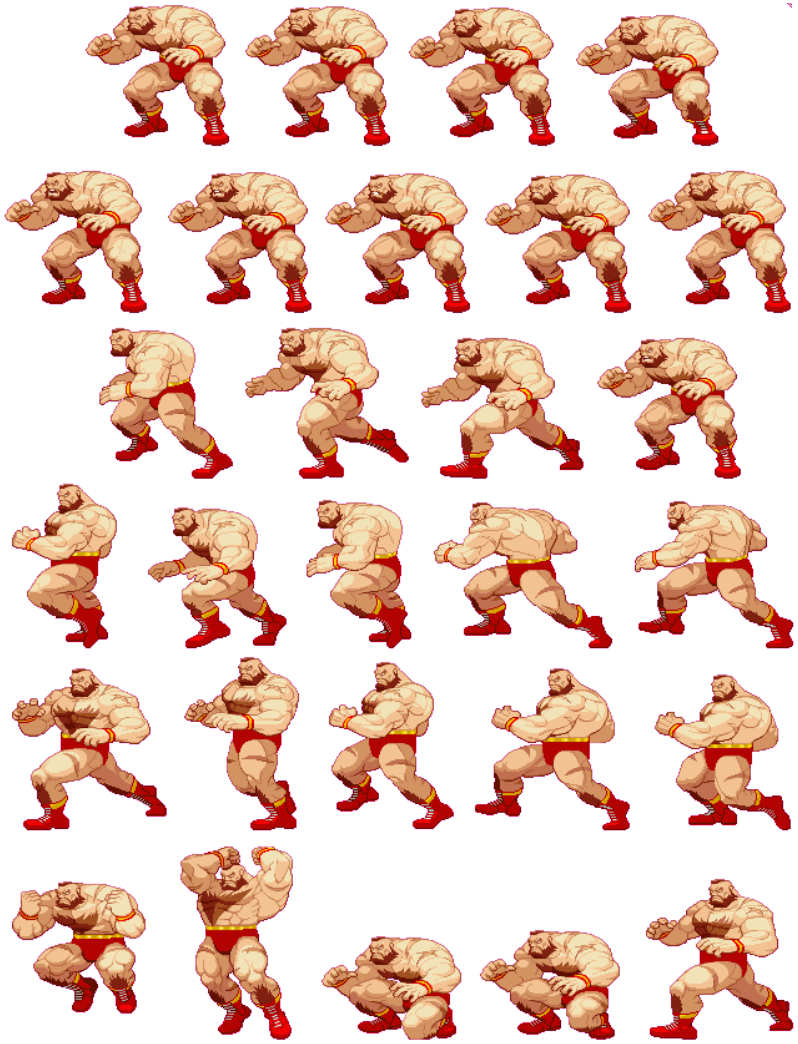


Utilisation de sprites

- Image finale

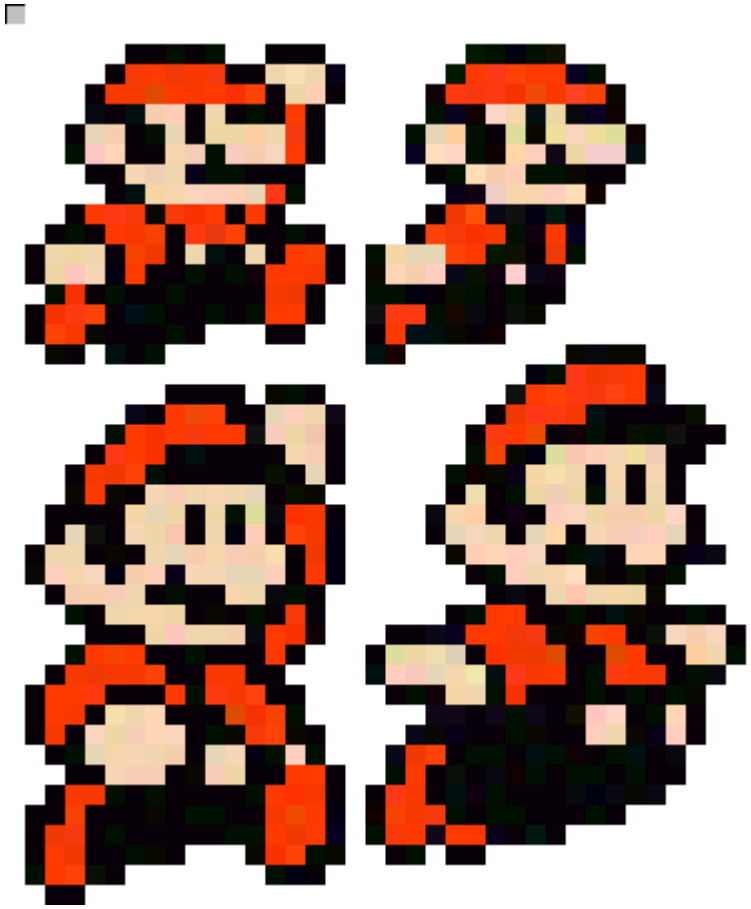
Extraire des sprites

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



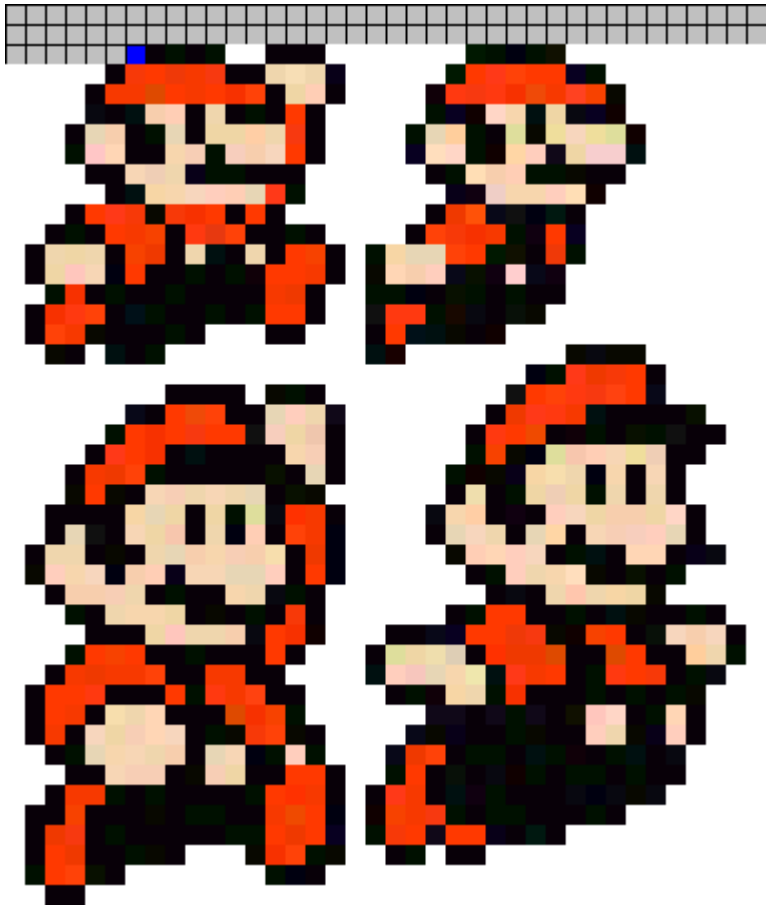
Extraction de zones connexes

- Partir d'un coin



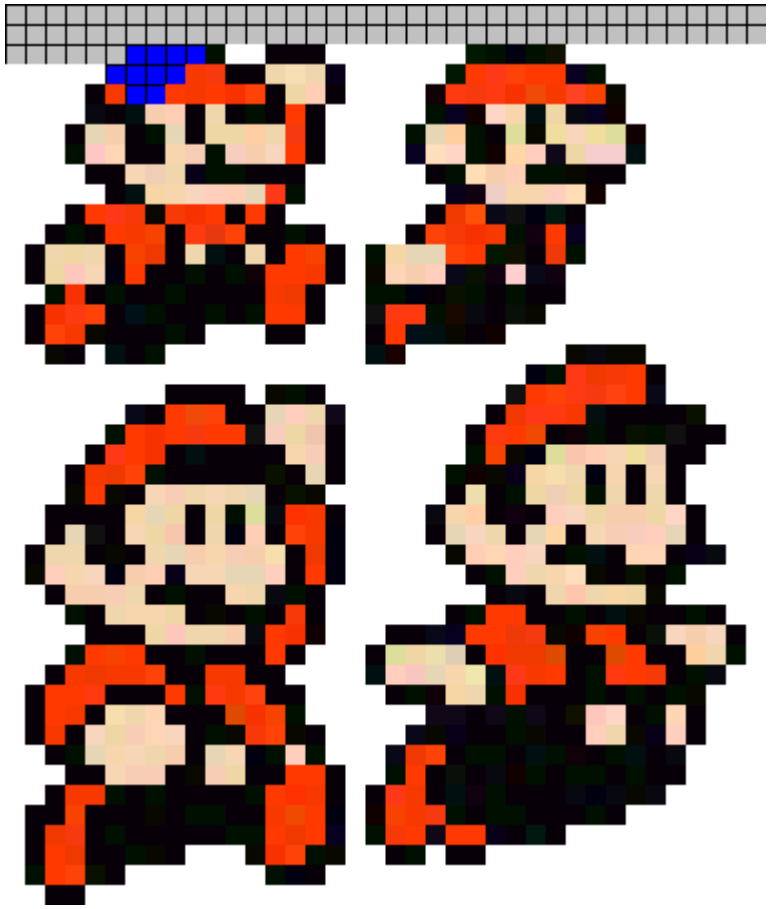
Extraction de zones connexes

- Partir d'un coin



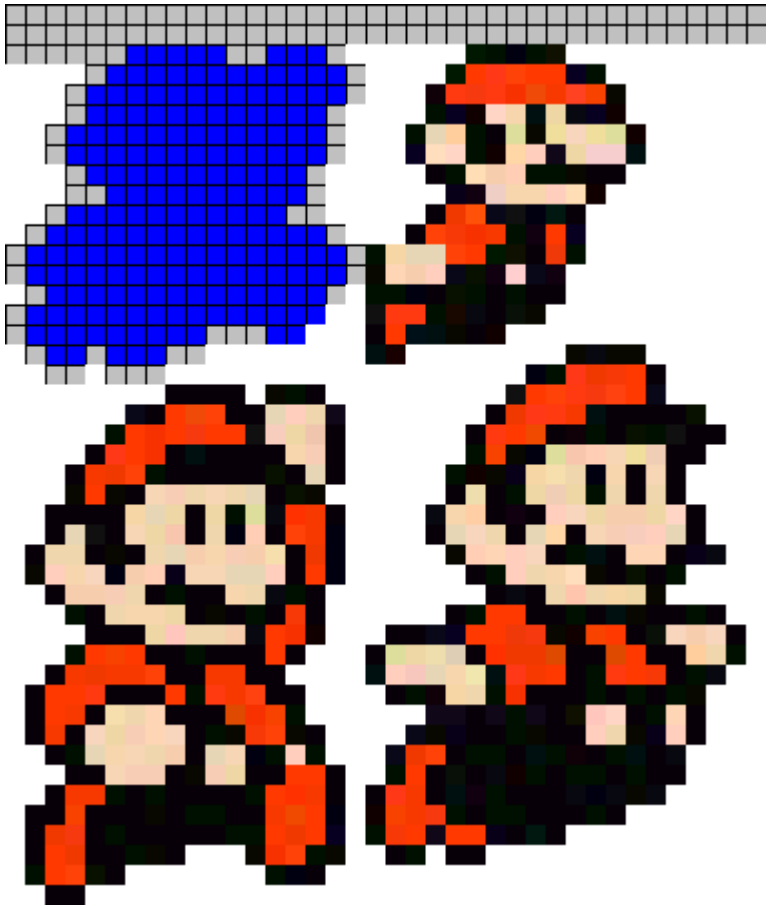
Extraction de zones connexes

- Partir d'un coin



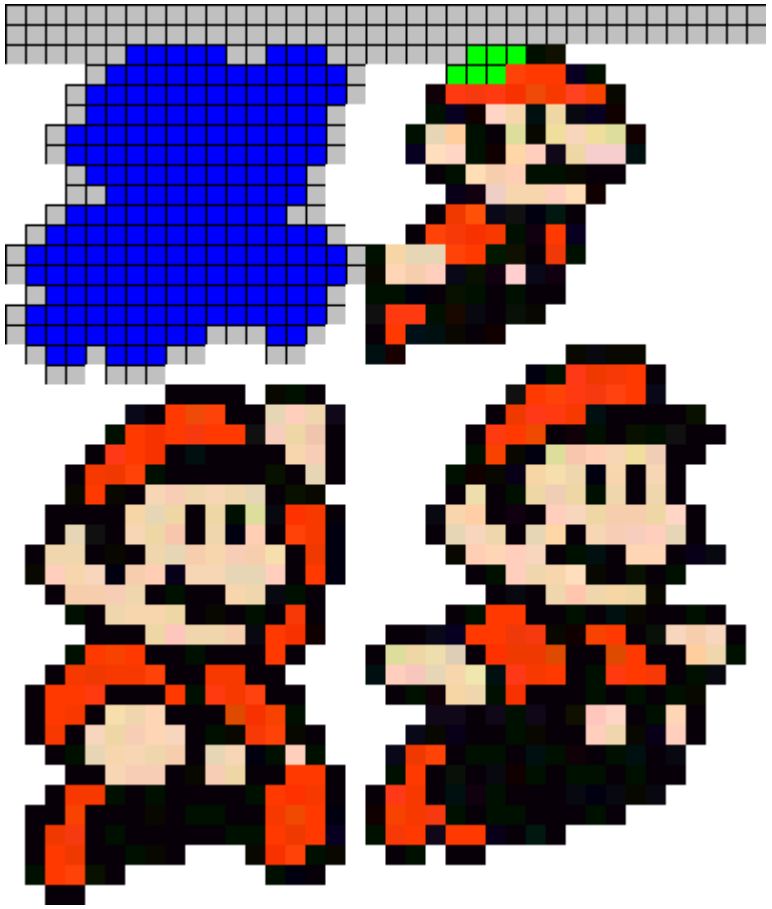
Extraction de zones connexes

- Partir d'un coin



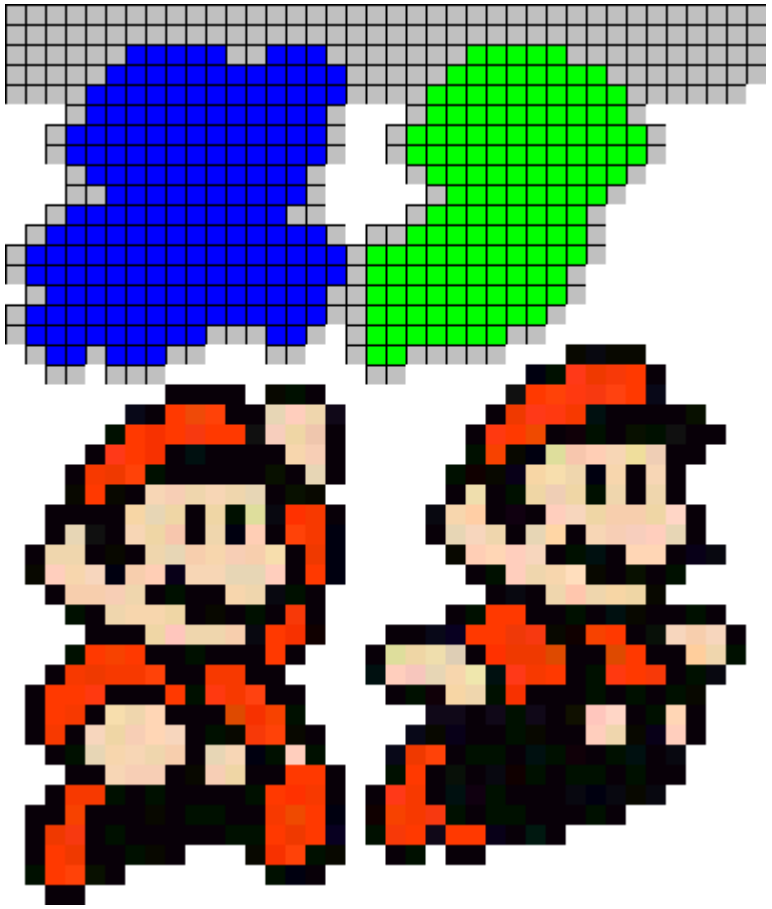
Extraction de zones connexes

- Partir d'un coin



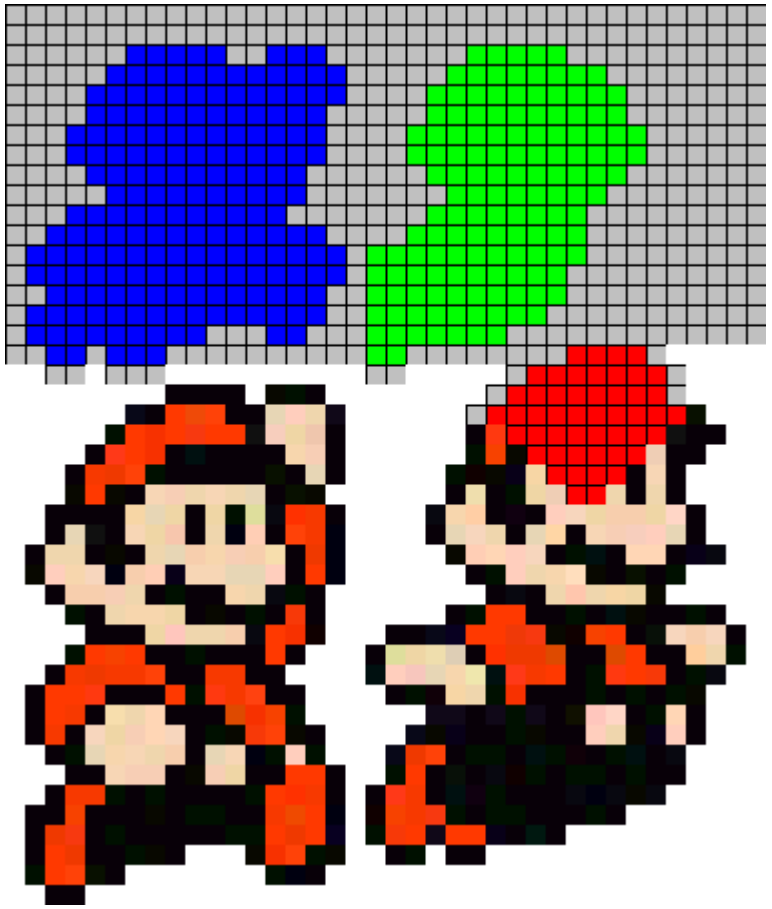
Extraction de zones connexes

- Partir d'un coin



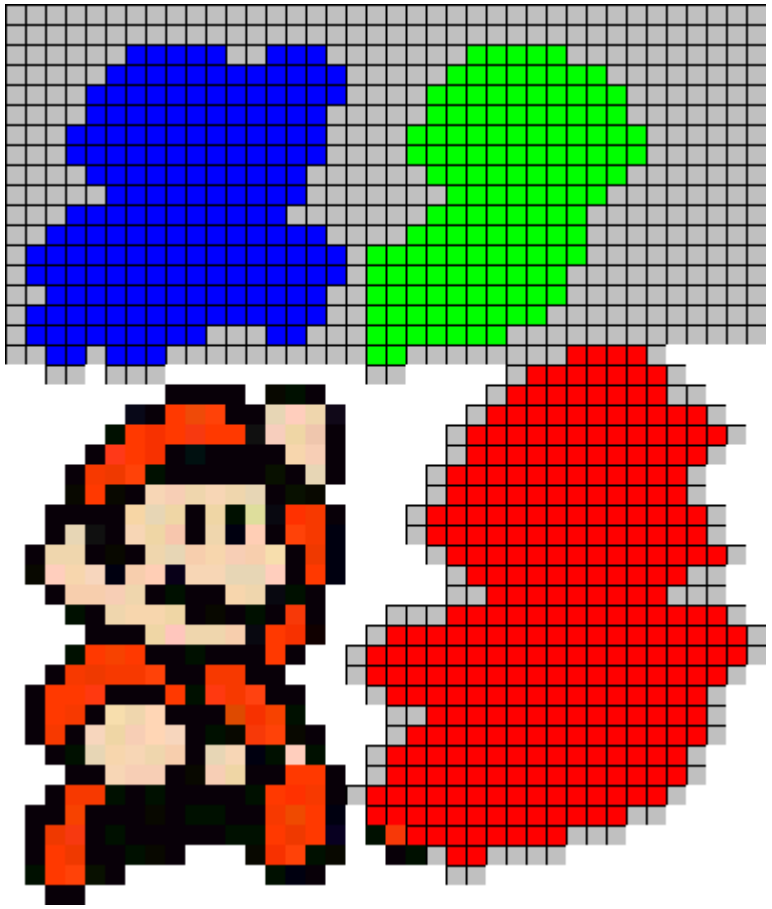
Extraction de zones connexes

- Partir d'un coin



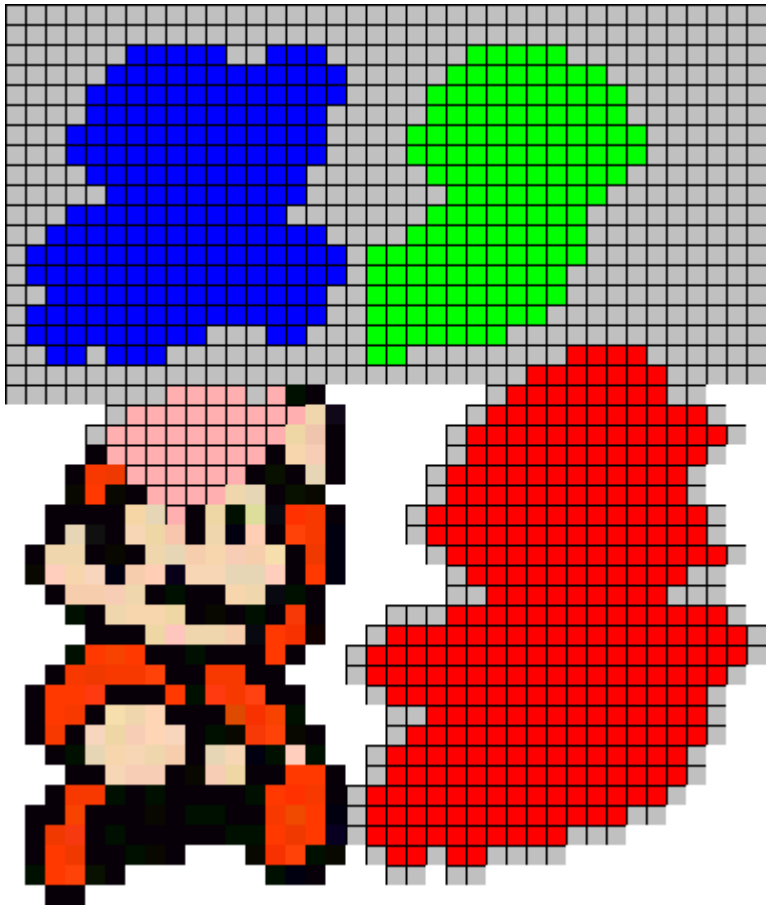
Extraction de zones connexes

- Partir d'un coin



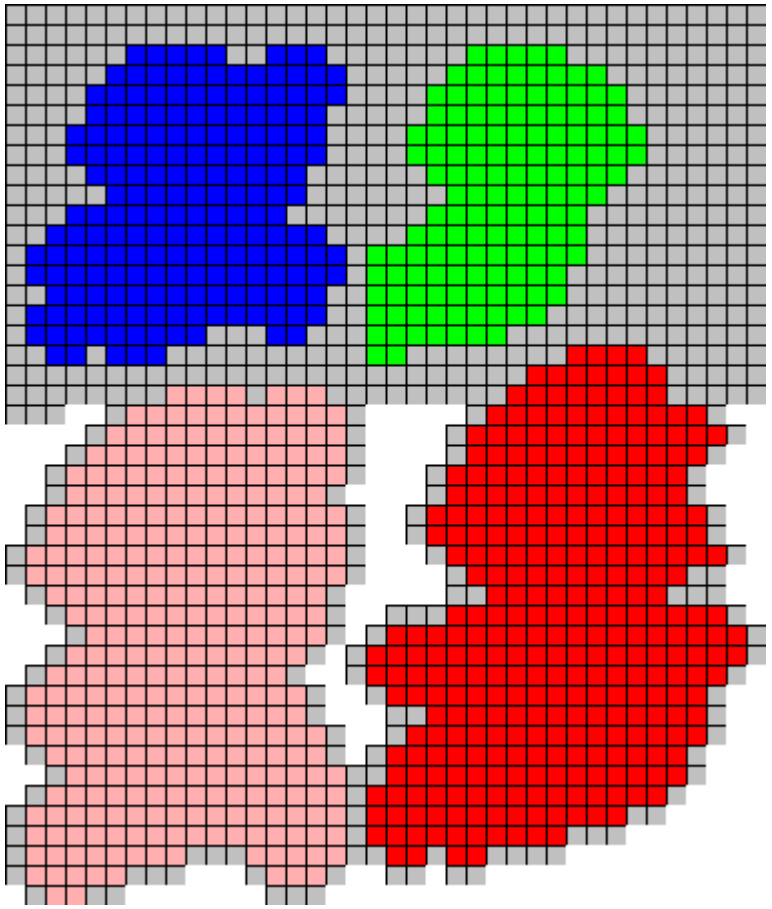
Extraction de zones connexes

- Partir d'un coin



Extraction de zones connexes

- Partir d'un coin

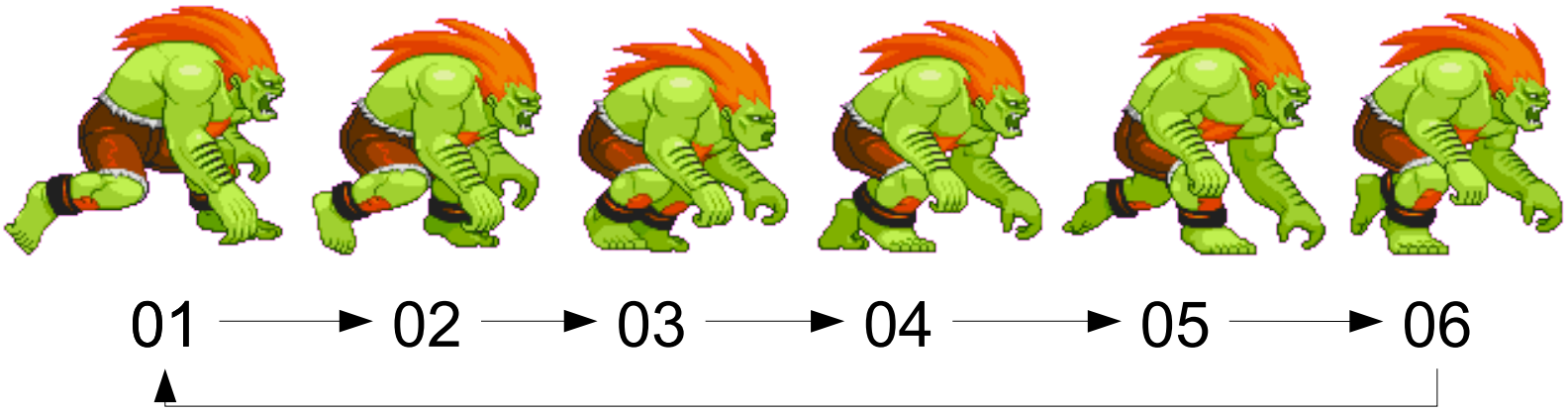


Démonstration Partie 8

Ajout sprite

Animation

- Changer sprite fonction du temps



- Pas un par frame
  - Faire un modulo
  - num = (it/10)%6+1

Animation

Animation

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

Animation (exemple)

- 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

Animation (exemple)

Int numero
String comportement

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

Animation (exemple)

Int numero
String comportement

Comportement

Marche

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

Poing

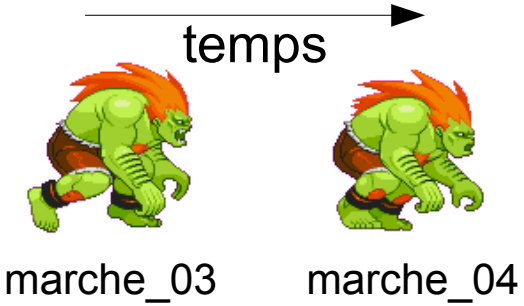
fonction numéro animation 1 → 2 → 2 → 3 → 3

Pied

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

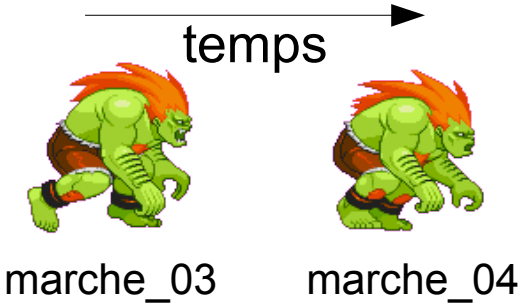
Animation (3 méthodes)

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

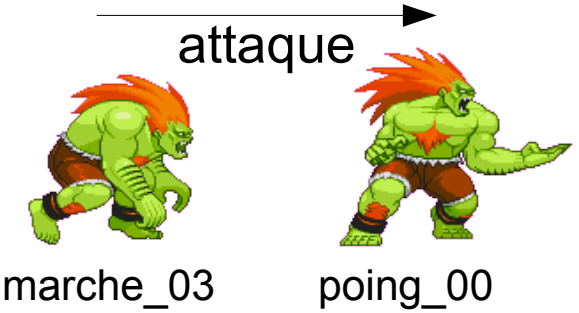


Animation (3 méthodes)

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

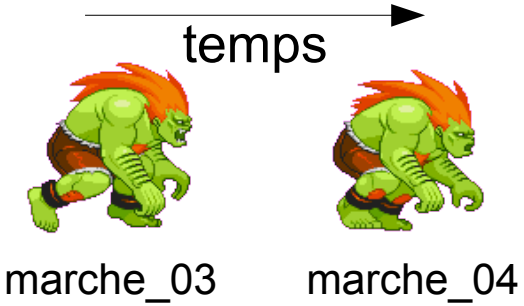


- changeCpt(int cpt)
  - Change comportement
  - Initialise numéro

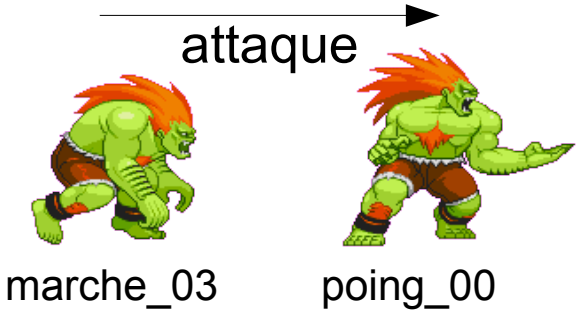


Animation (3 méthodes)

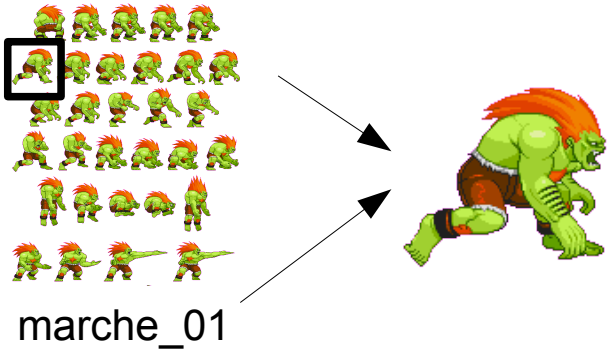
- anime()
  - Modifie numéro de frame
  - Gère temps et cycles



- changeCpt(int cpt)
  - Change comportement
  - Initialise numéro



- afficheSprite(int x, int y)
  - Comportement + numéro
  - Retourne image extraite



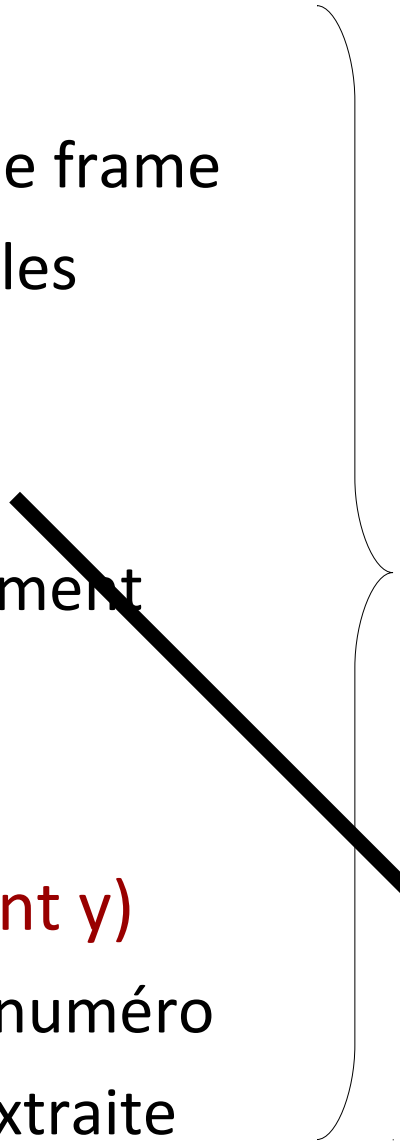
Animation (3 méthodes)

- 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

Animation (3 méthodes)

- 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

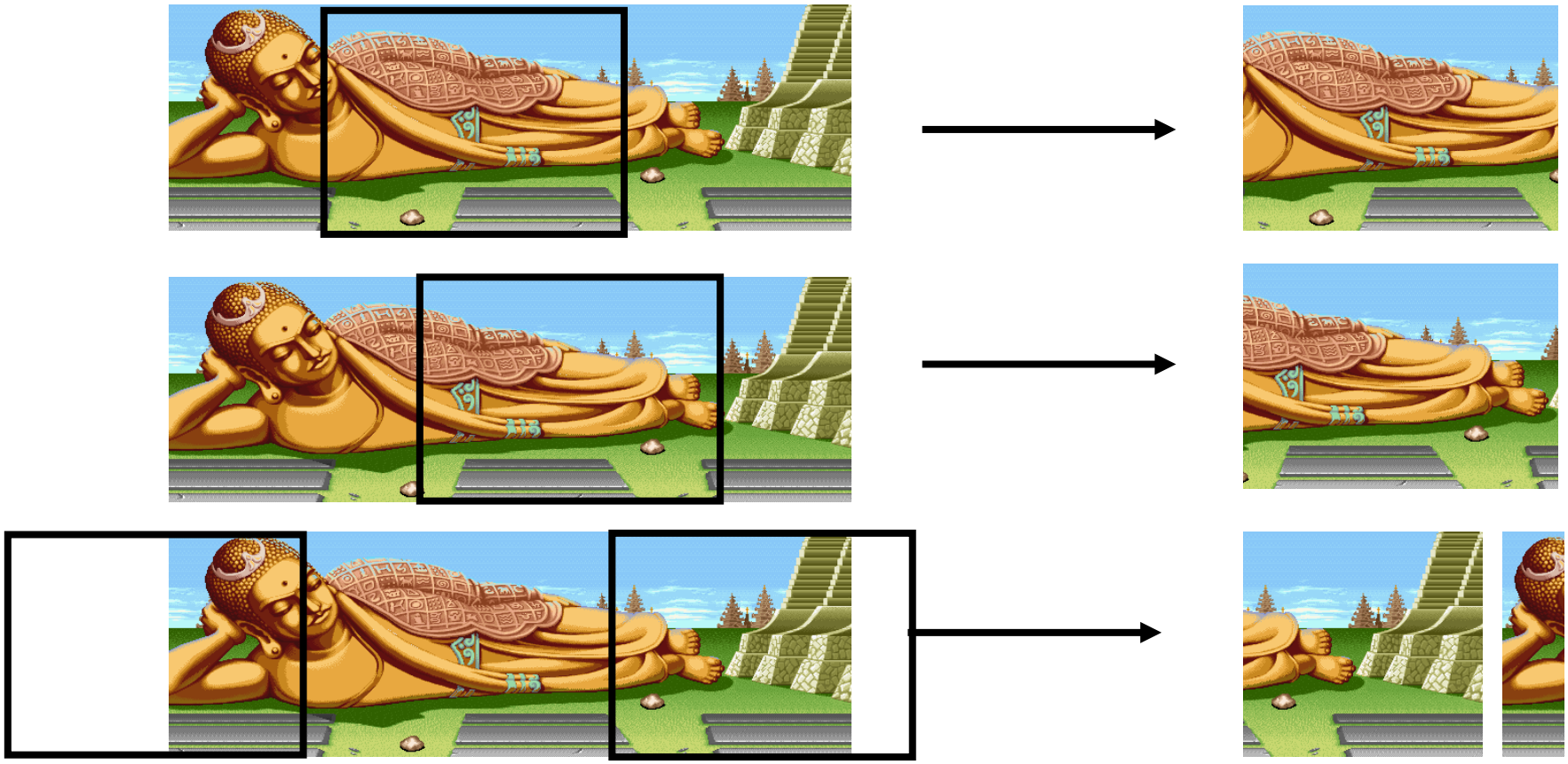
Démonstration 09

Animation de sprite

Blanka, Plateforme

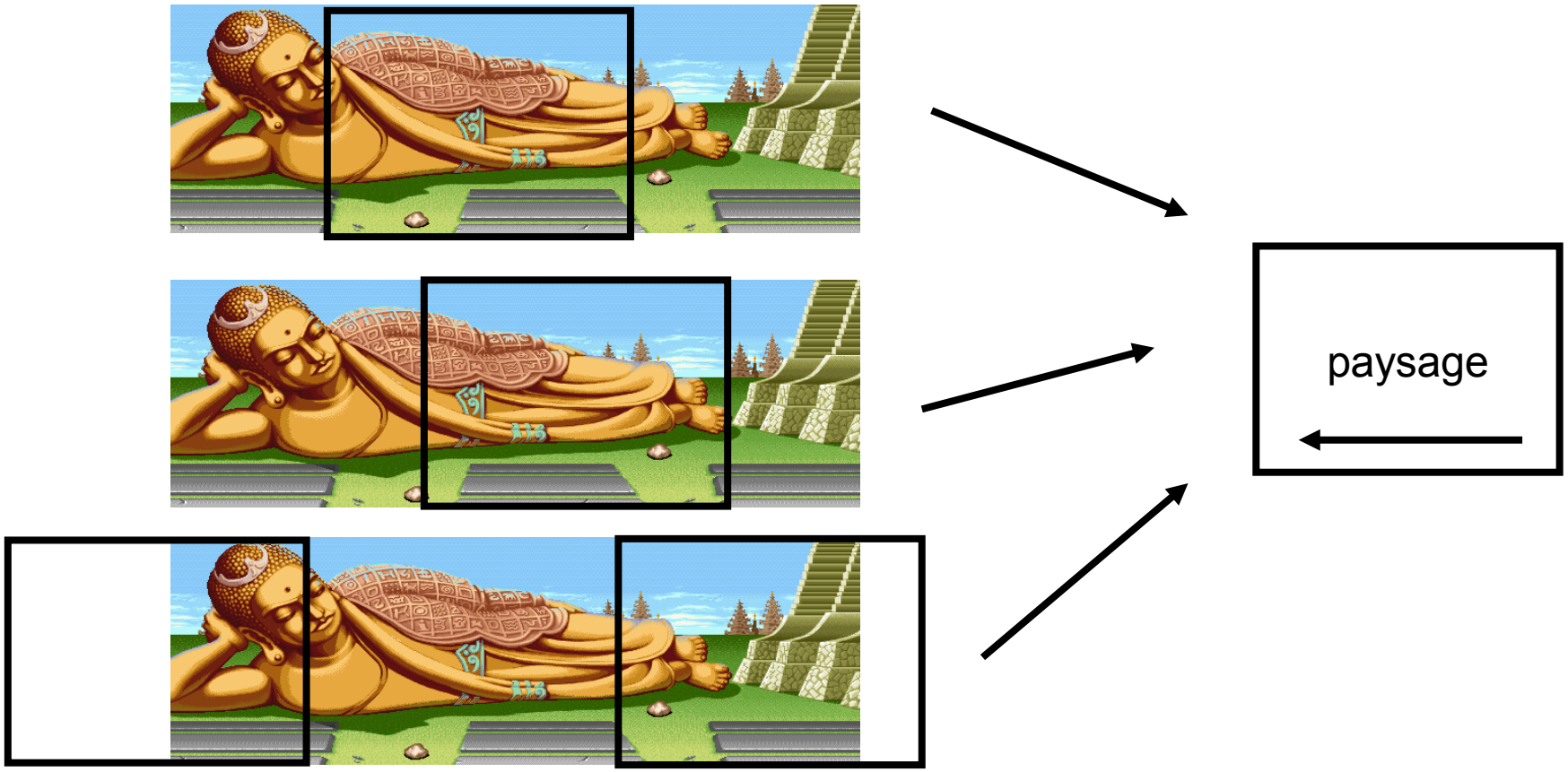
Quelques exemples

- Scrolling
  - Défiler un décor copie morceaux image



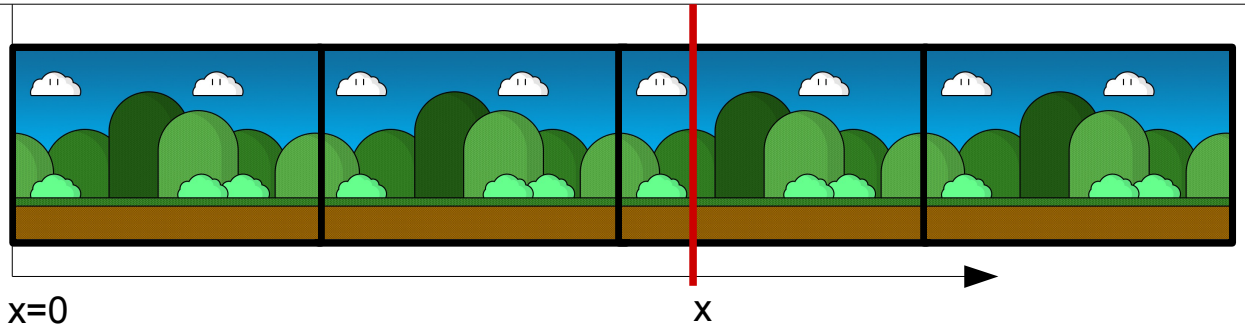
Quelques exemples

- Scrolling
  - Défiler un décor copie morceaux image

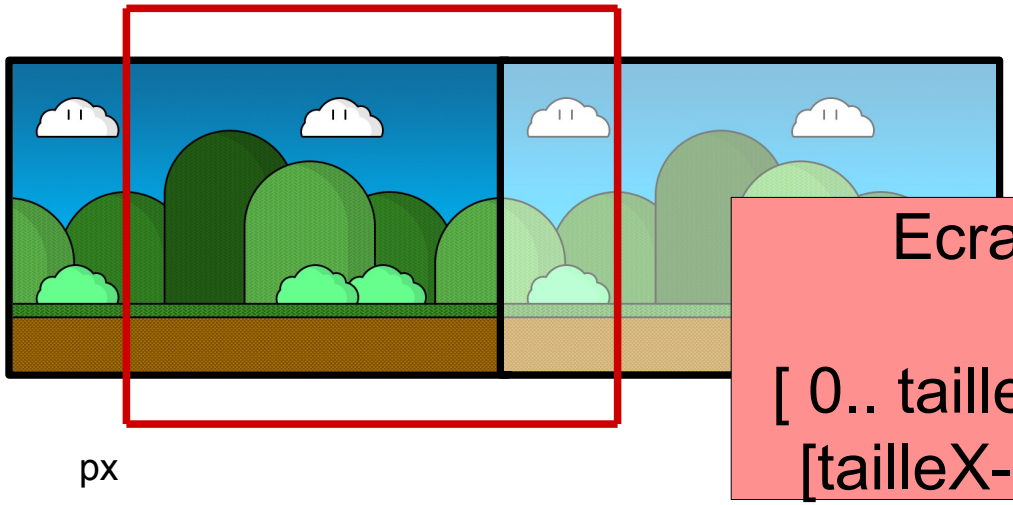


Code Scrolling

- Scrolling



- Position de x dans image n
  - Px = (x % tailleX)

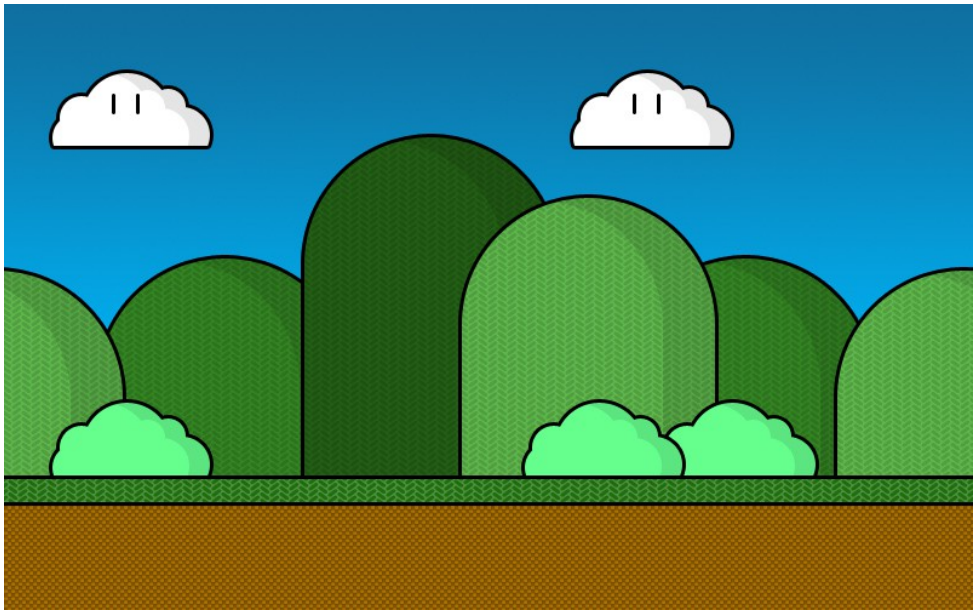


Ecran => image source

[0.. tailleX-px] => [px .. tailleX]
[tailleX-px .. tailleX] => [0 .. px]

Code Scrolling

- Scrolling



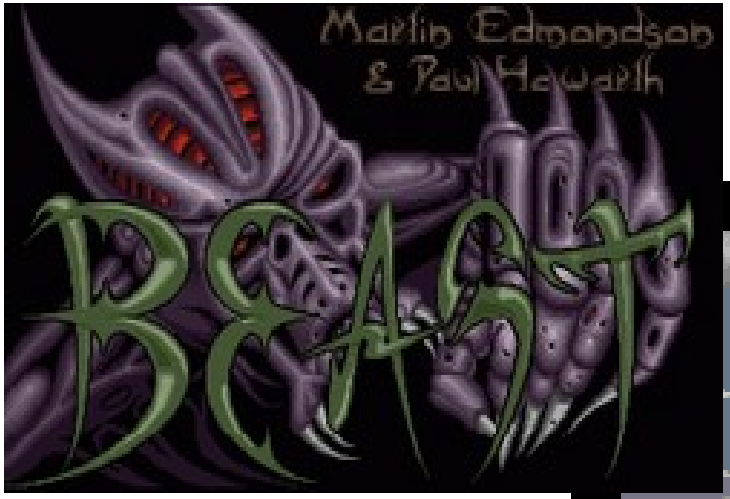
```
public void affiche(int x,Graphics g)
{
//on se ramene au repère du plan
x=(x%wx);

//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);

}
```

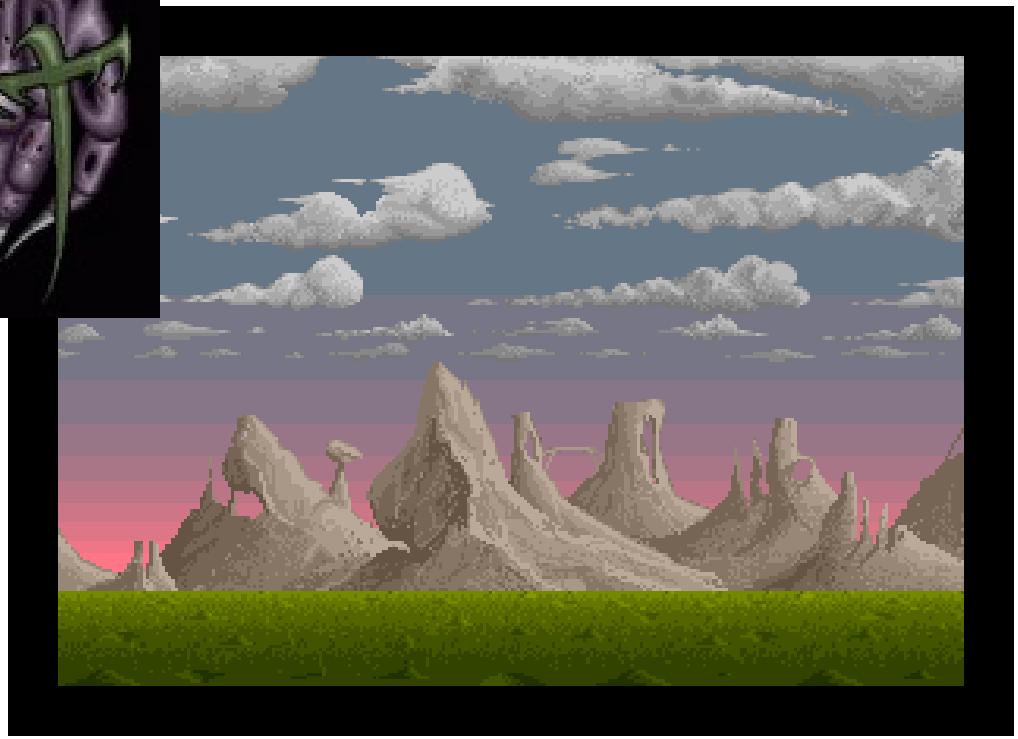
Démonstration

Scrolling

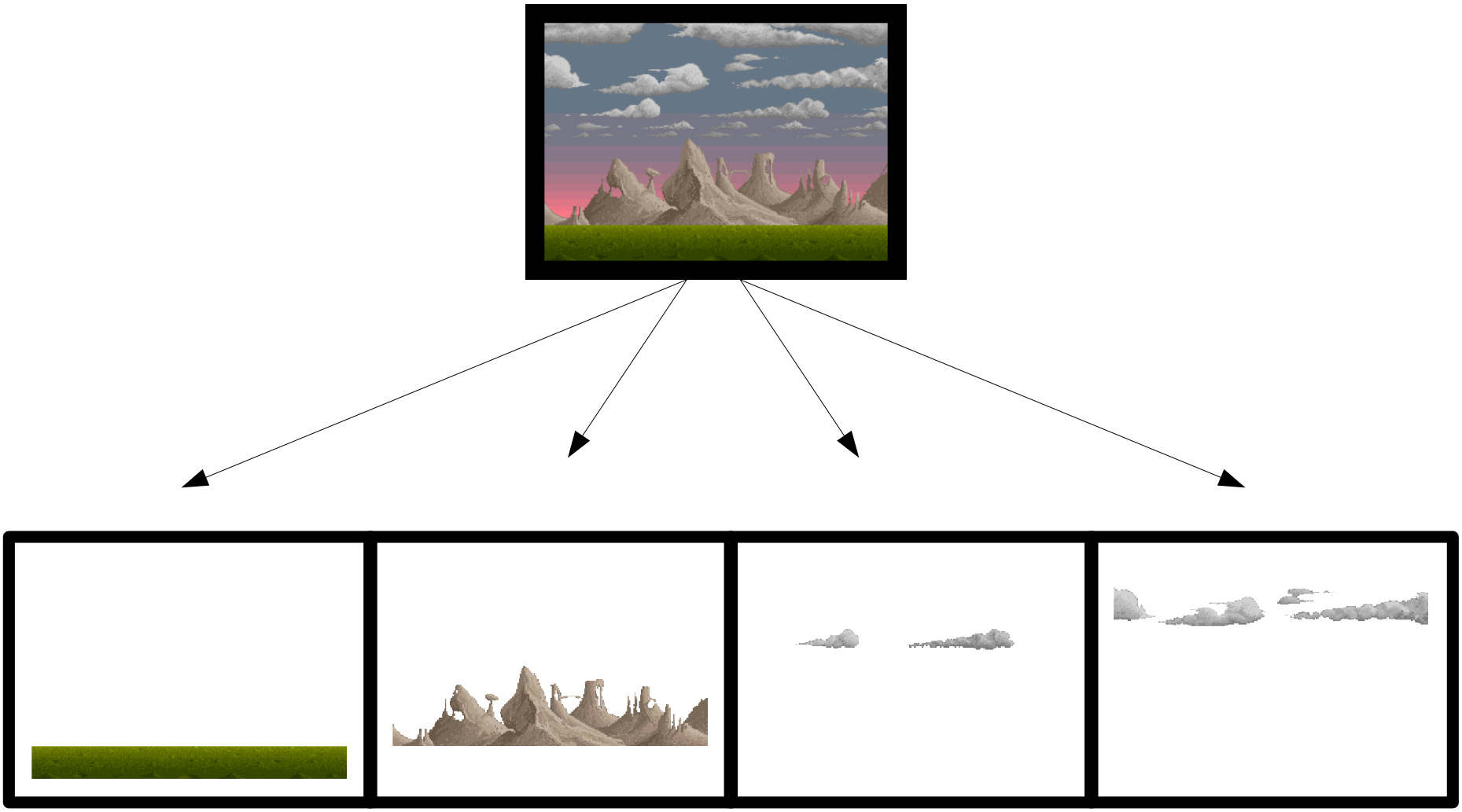
Défilement parallaxe

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

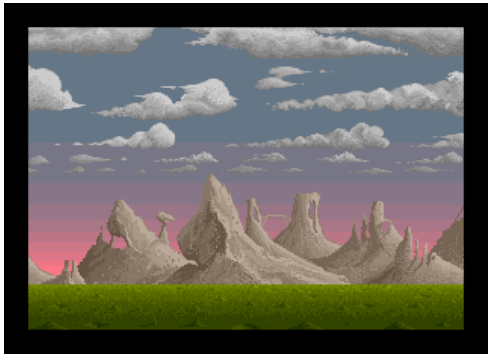




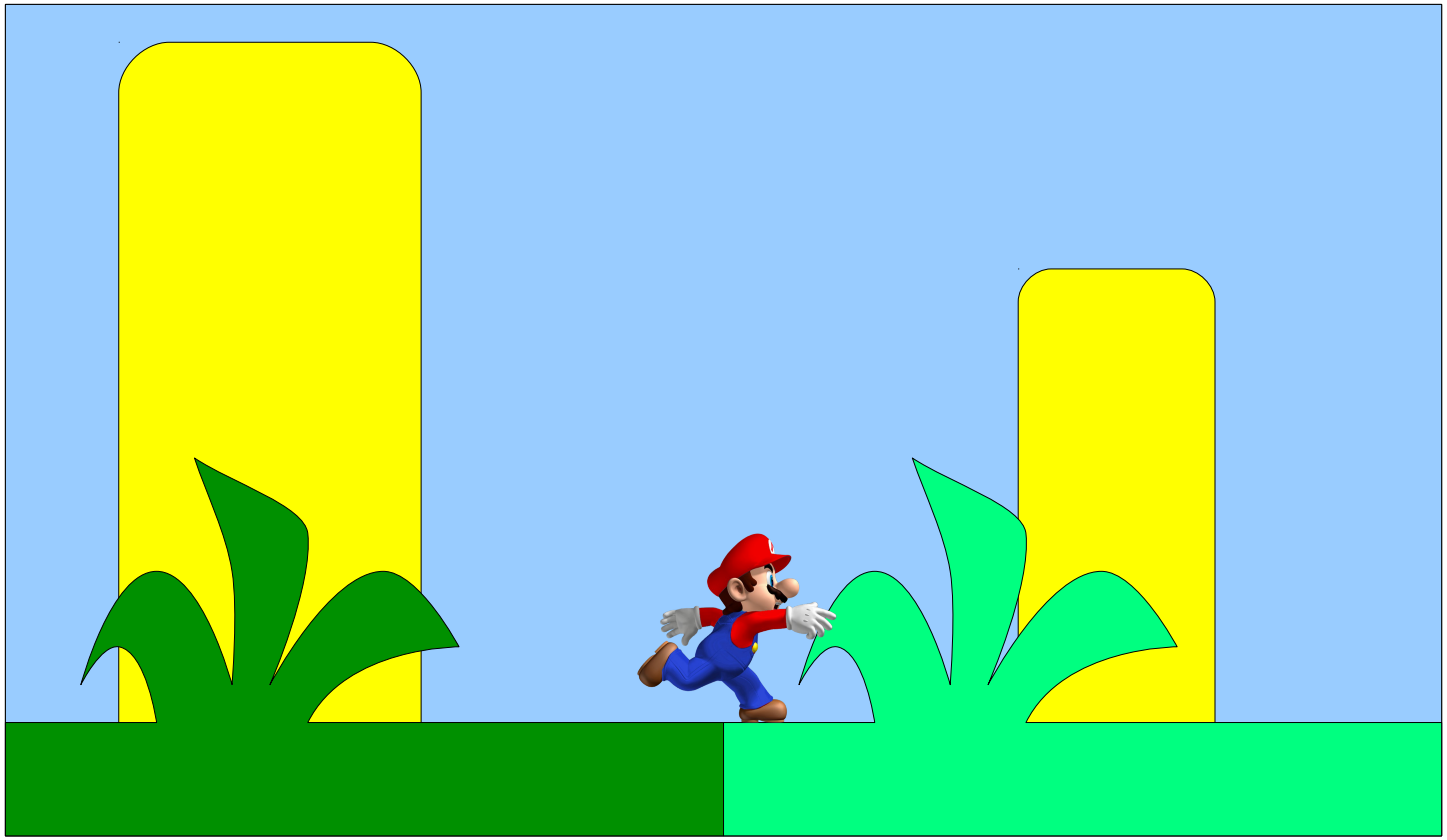
Décomposition en plans



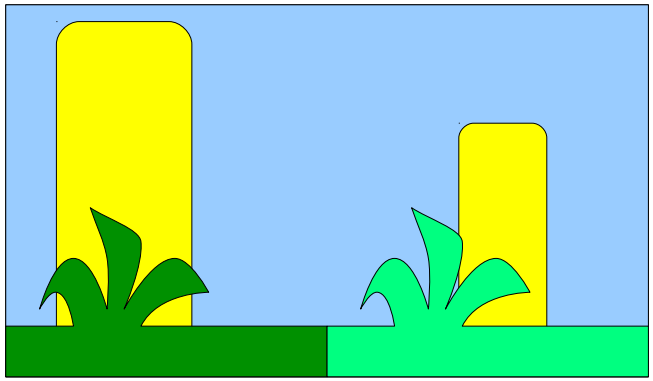
Décomposition en plans



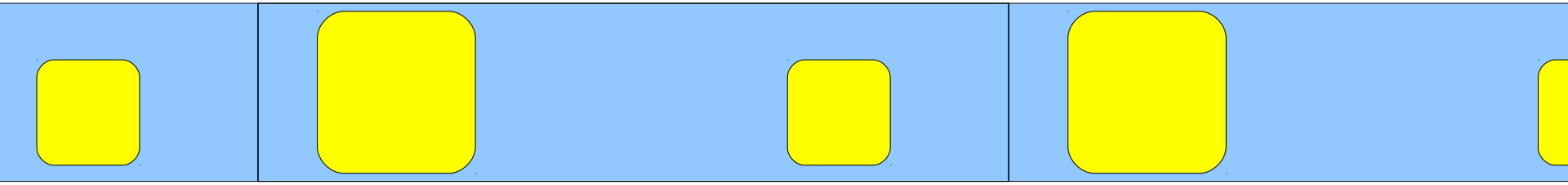
Défilement parallaxe

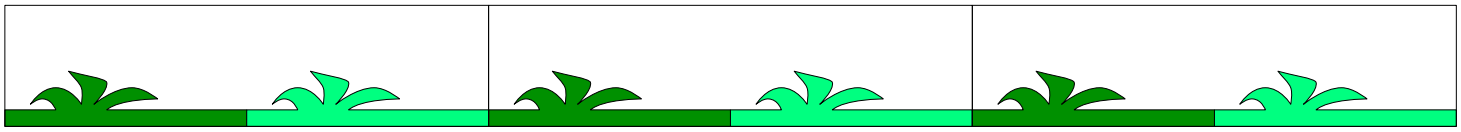


Défilement parallaxe



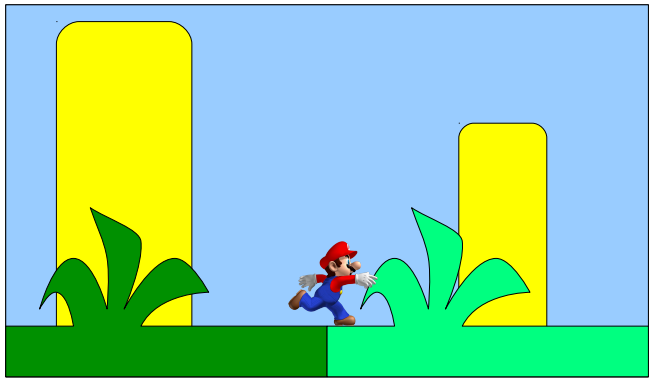
x2

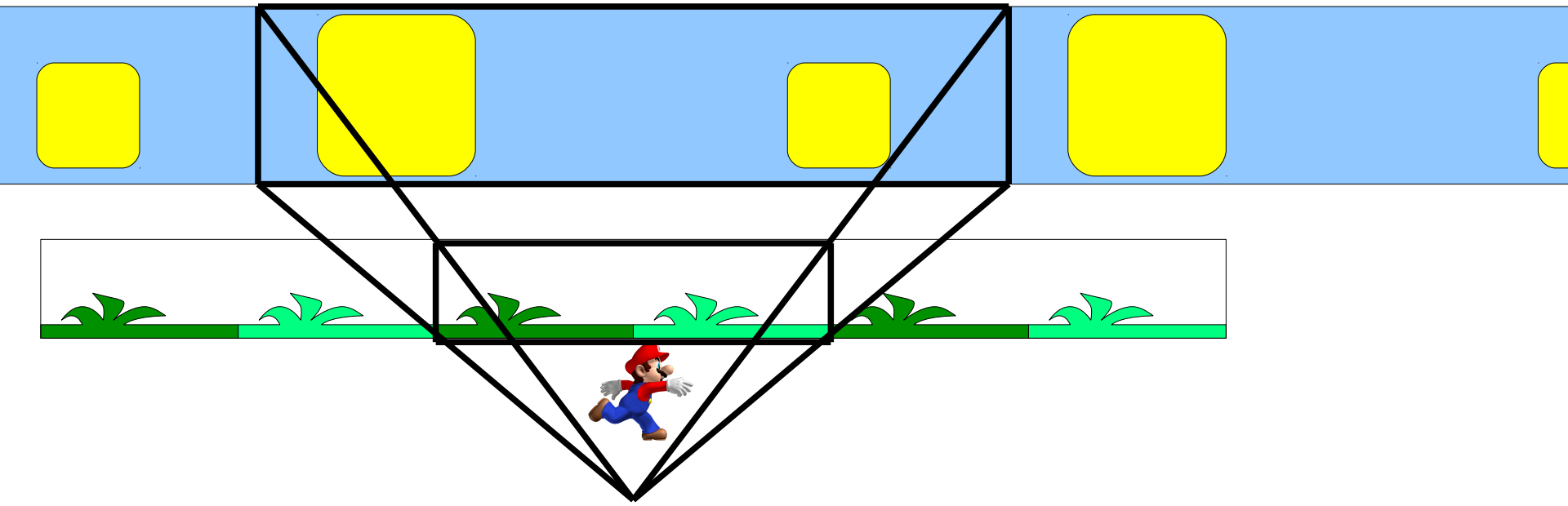




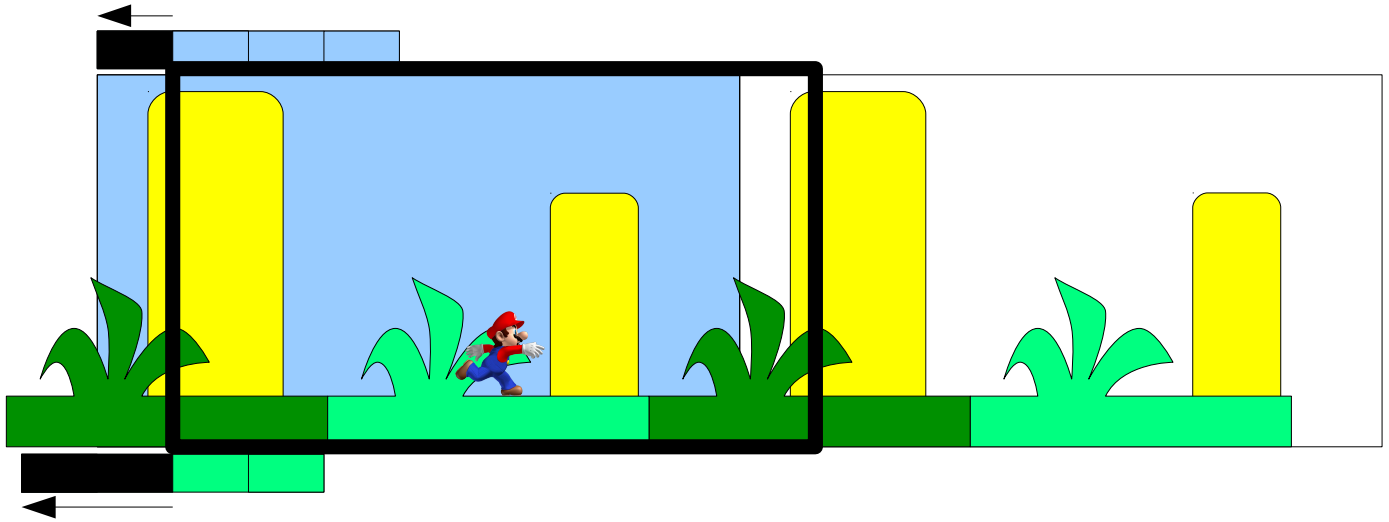
x1

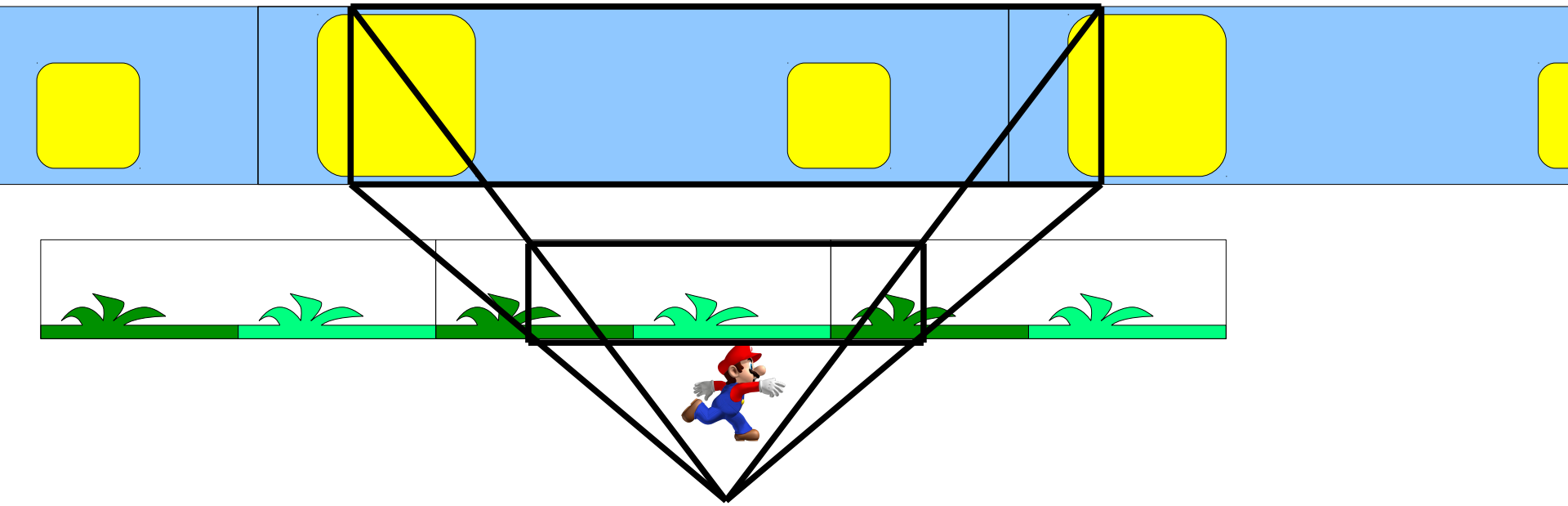
Défilement parallaxe



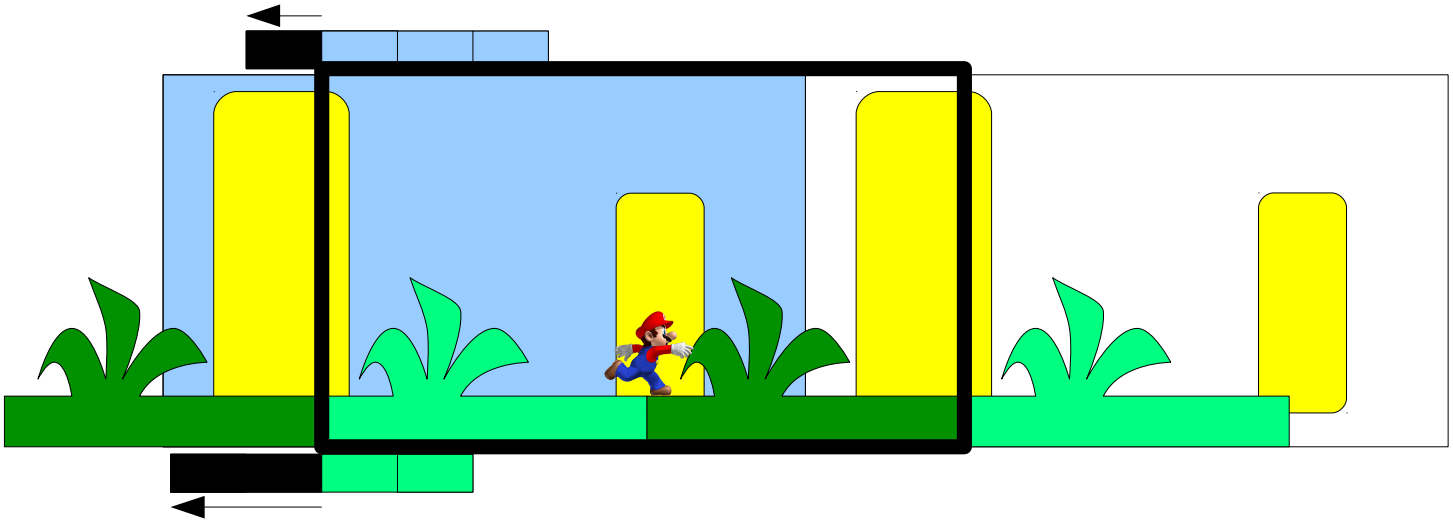


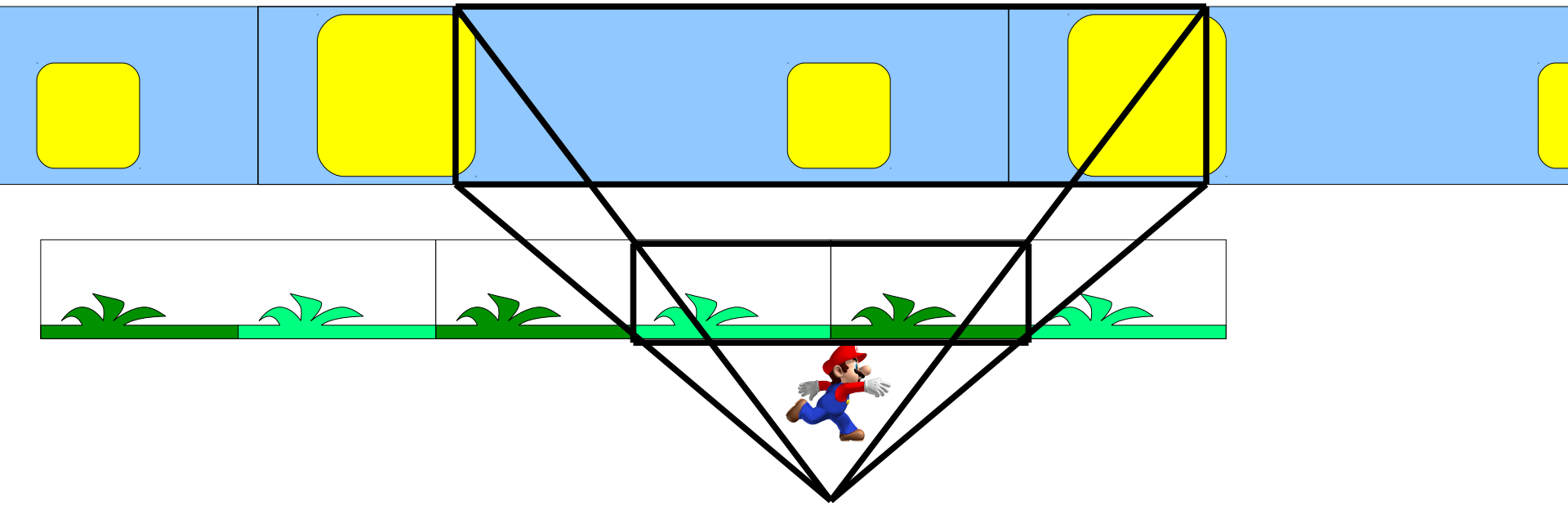
Défilement parallaxe



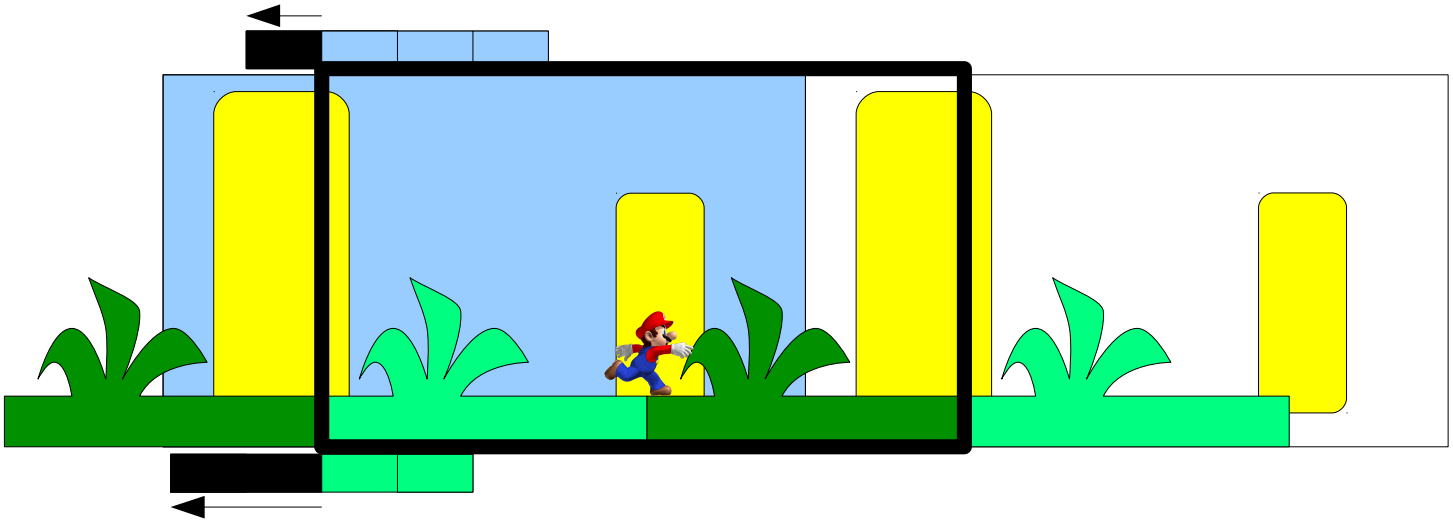


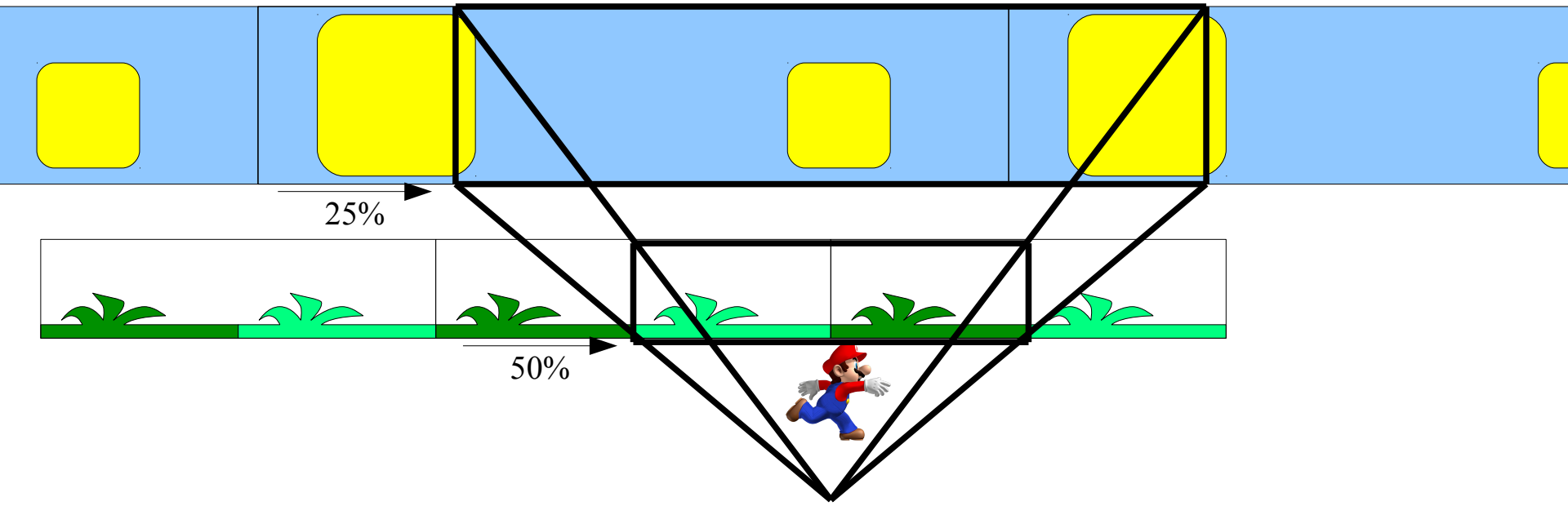
Défilement parallaxe



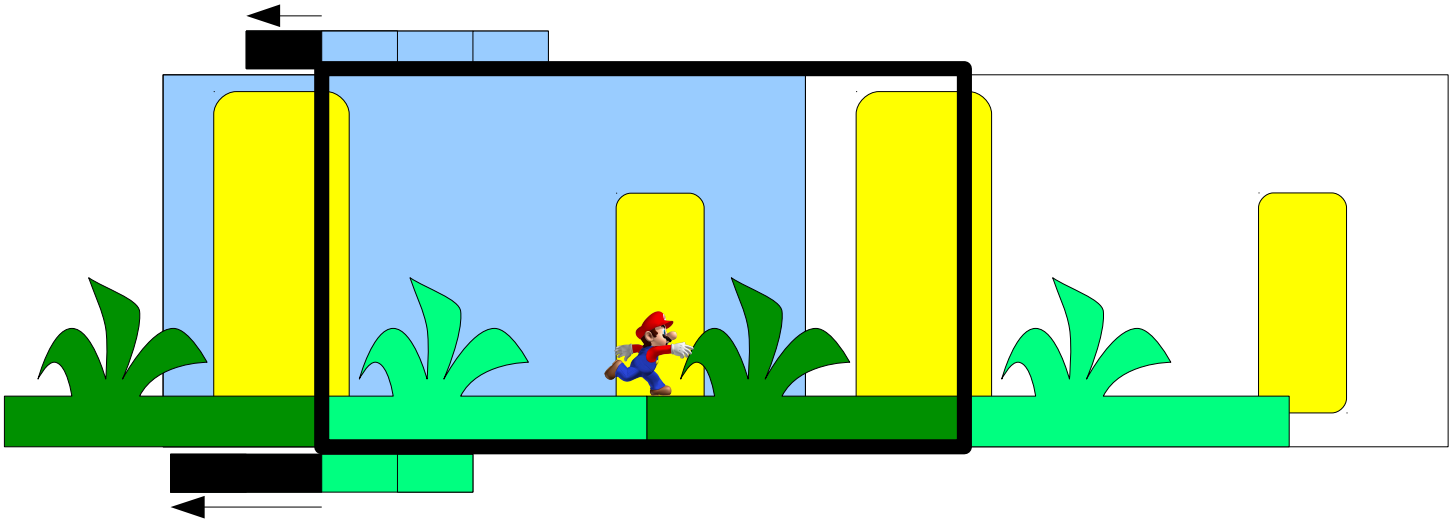


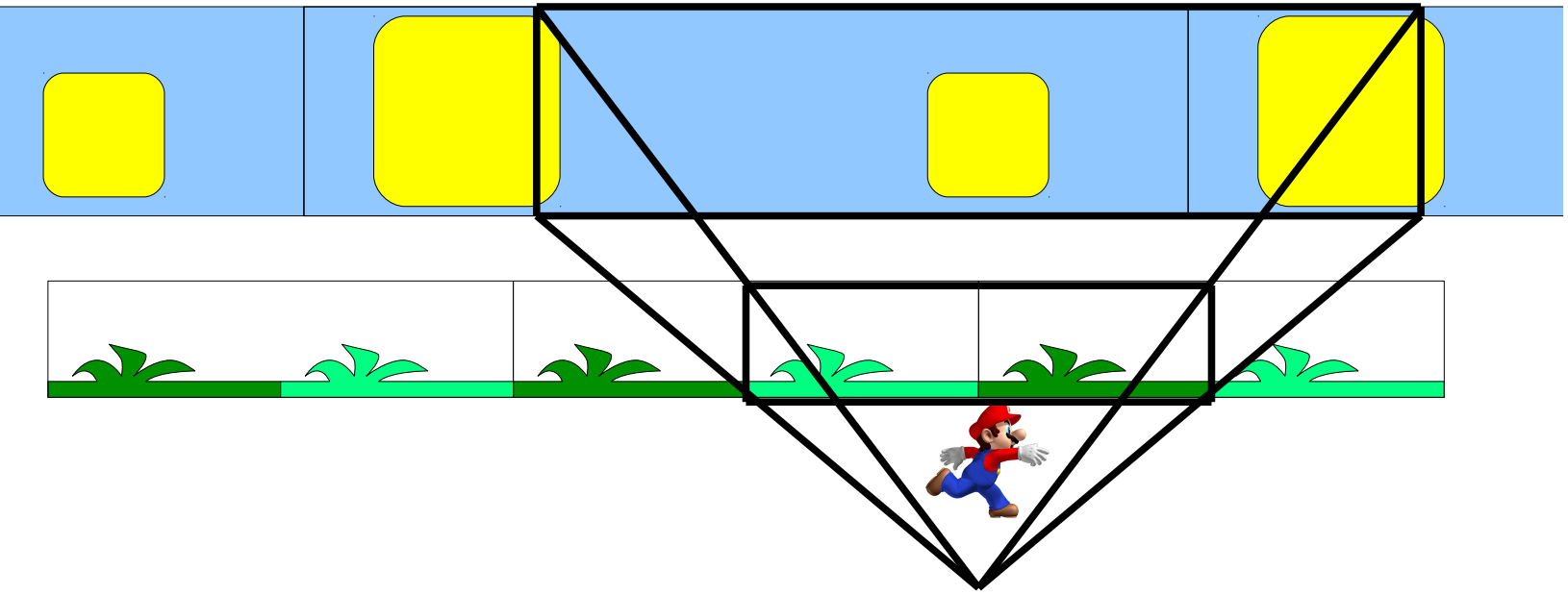
Défilement parallaxe



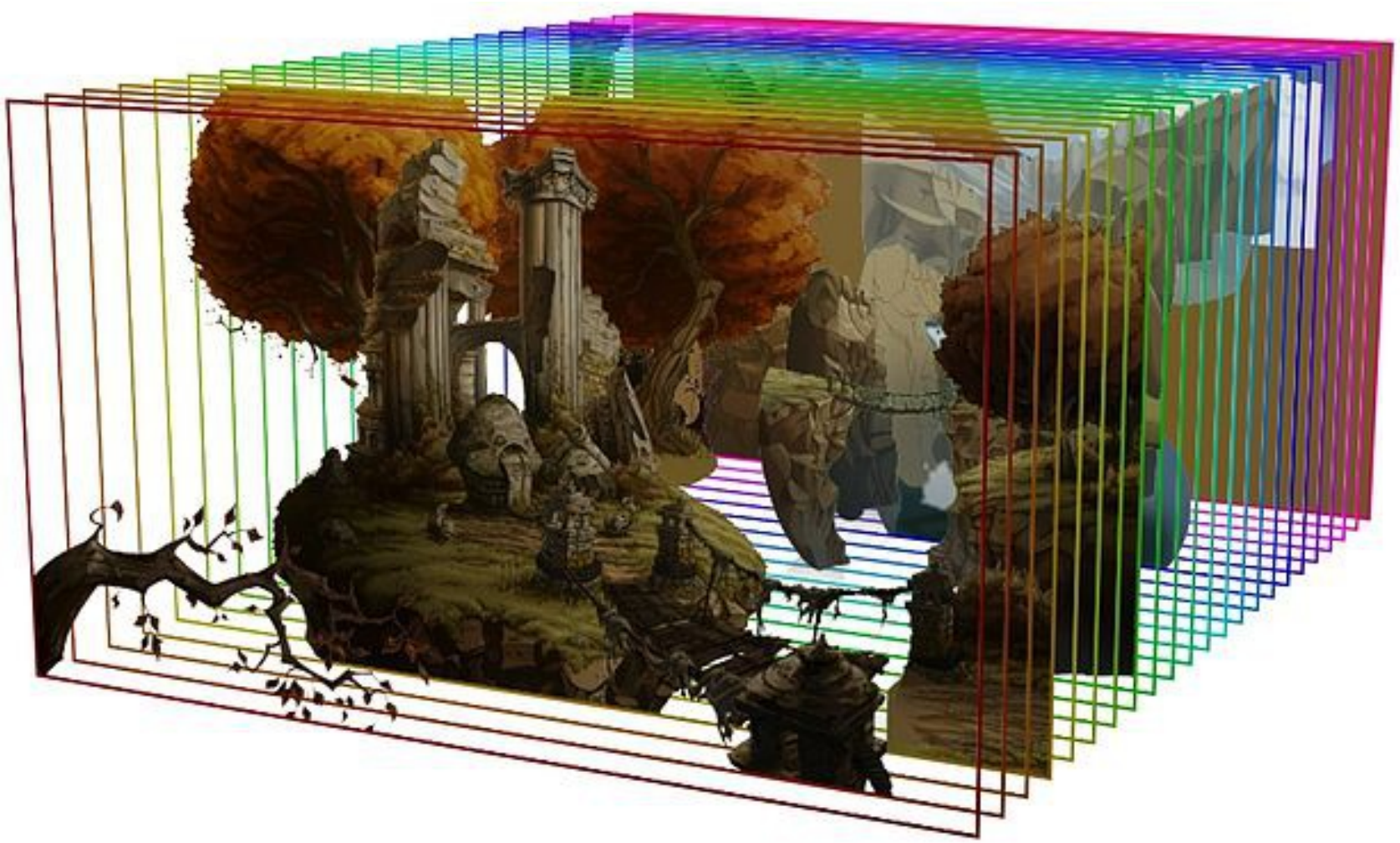


Défilement parallaxe





Parallaxe



http://ollytyler.blogspot.fr

Démonstration

Défilement Parallaxe

Repères

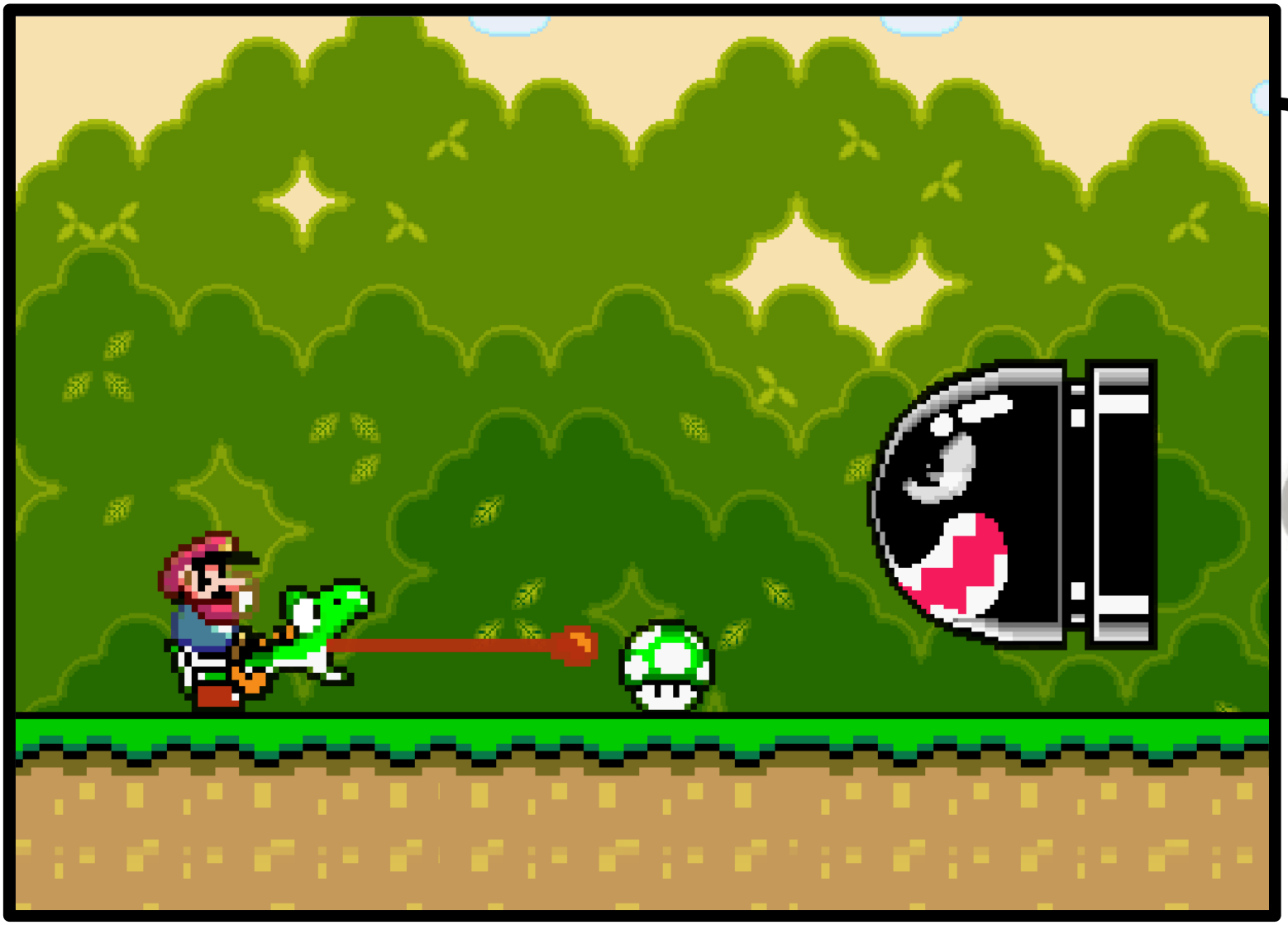
- Modifications des repères de jeu
  - Suivi vertical
  - Zoom
  - Changement vitesse

- Démonstrations

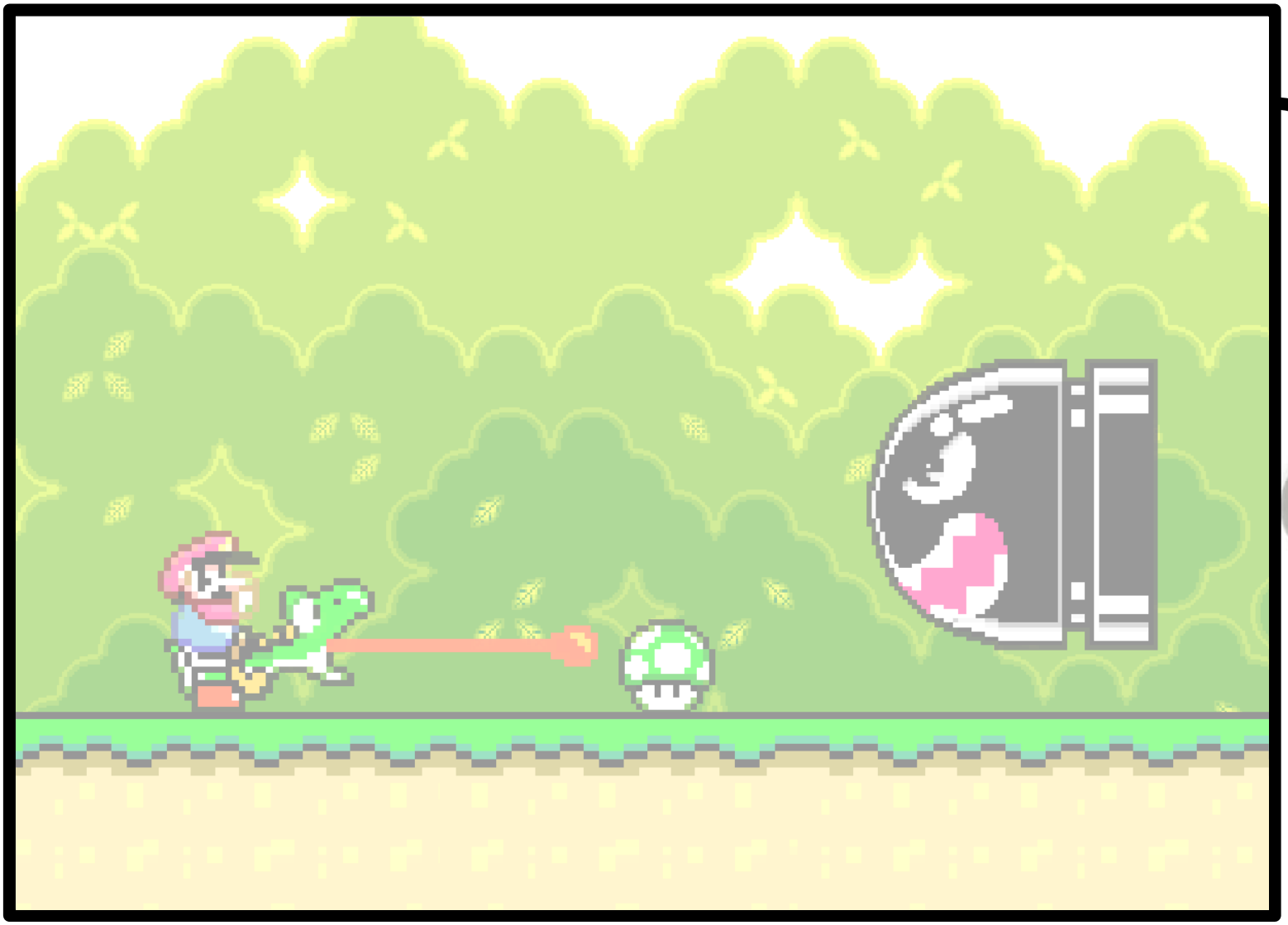
Réseau

- Utiliser les controleurs

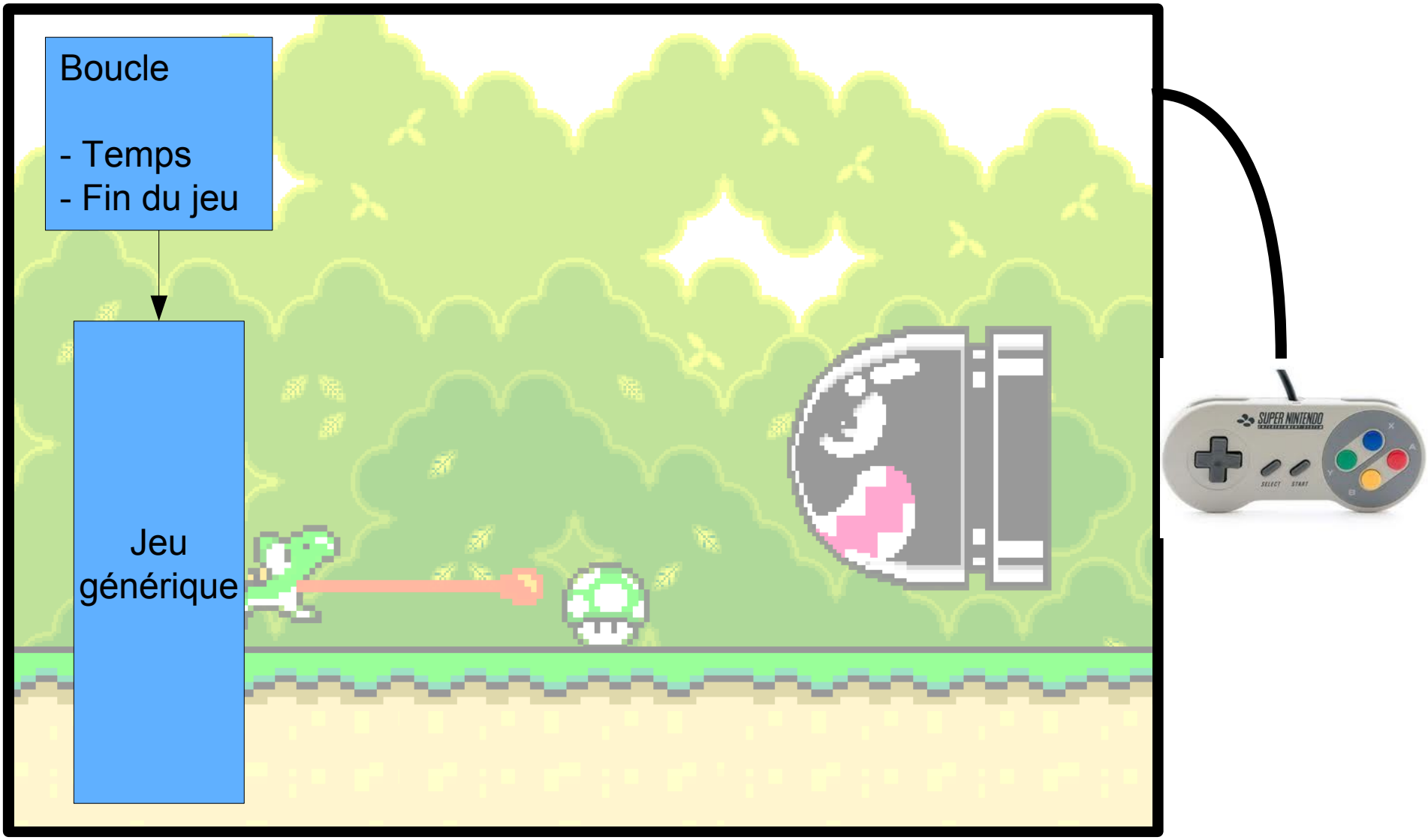
Bilan



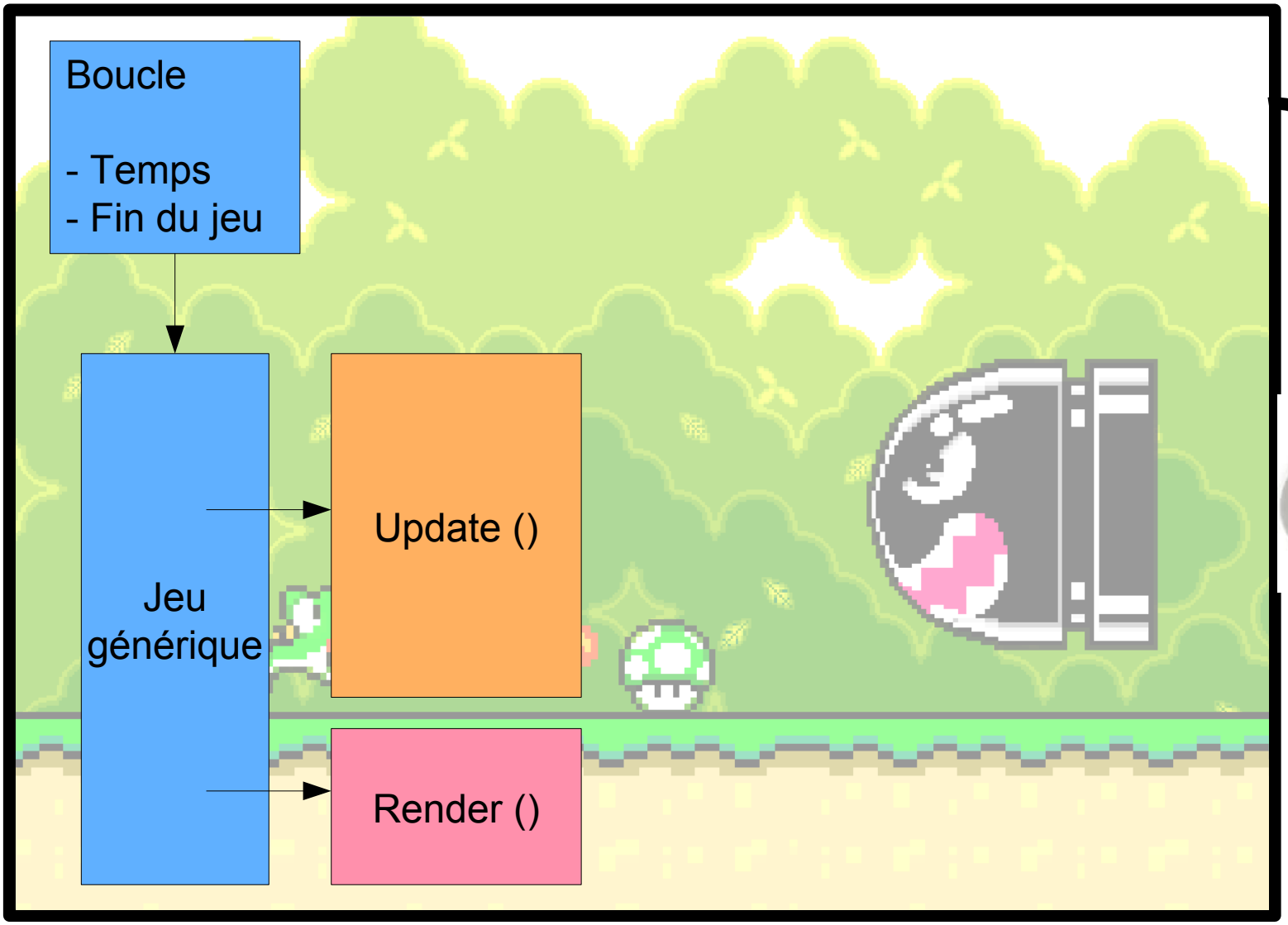
Bilan



Bilan

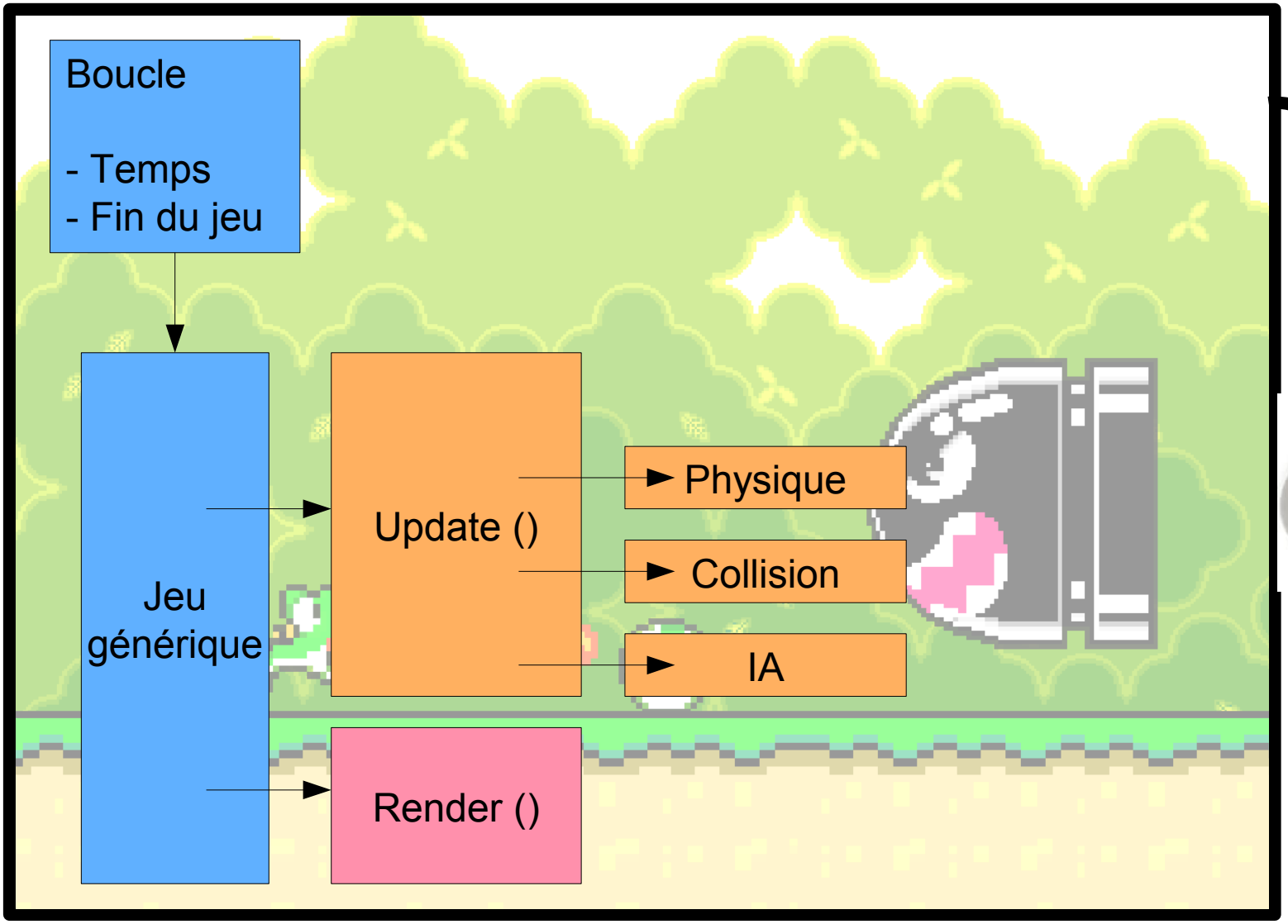


Bilan


Boucle
- Temps
- Fin du jeu
Jeu générique
Update ()
Render ()

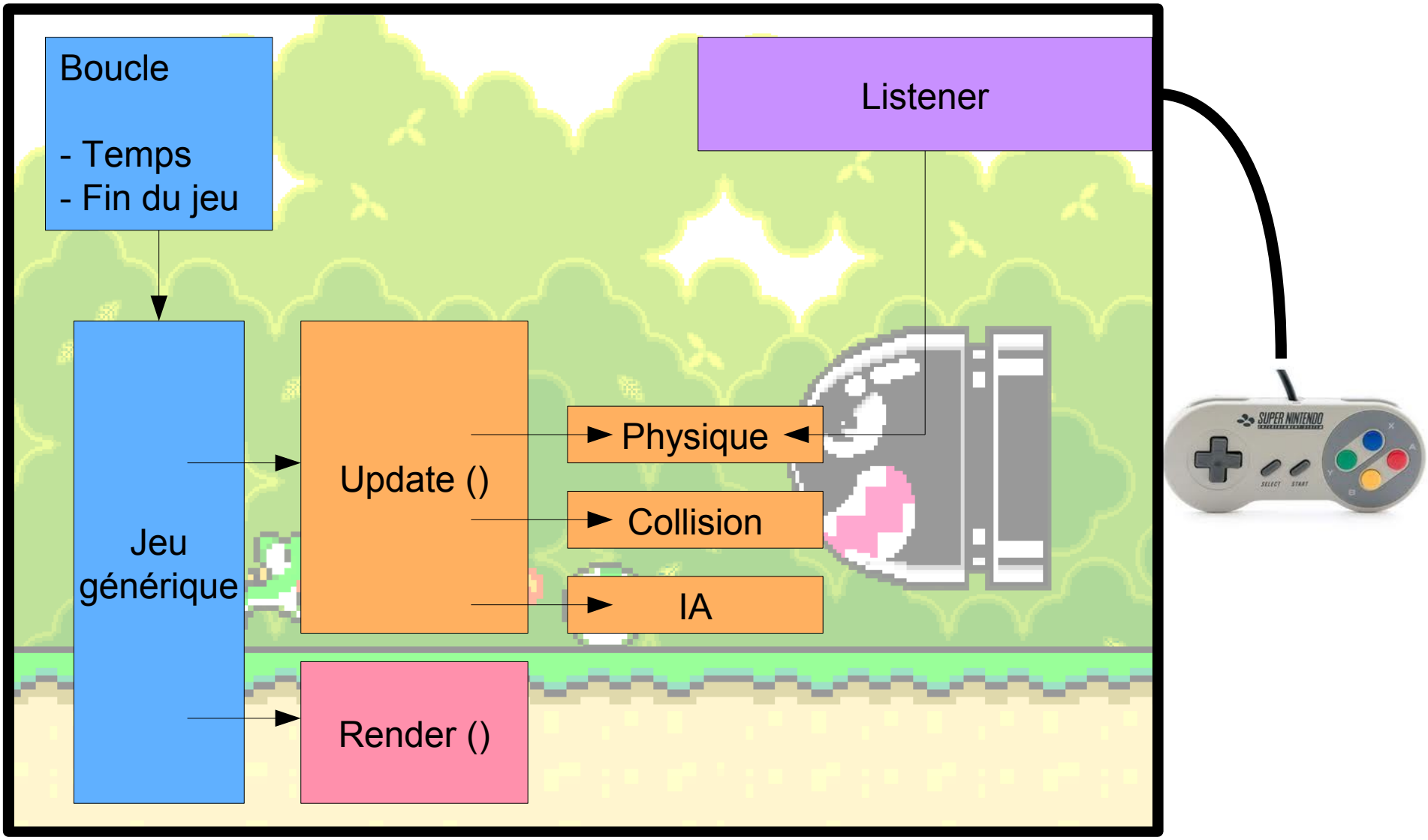

SUPER NINTENDO

Bilan

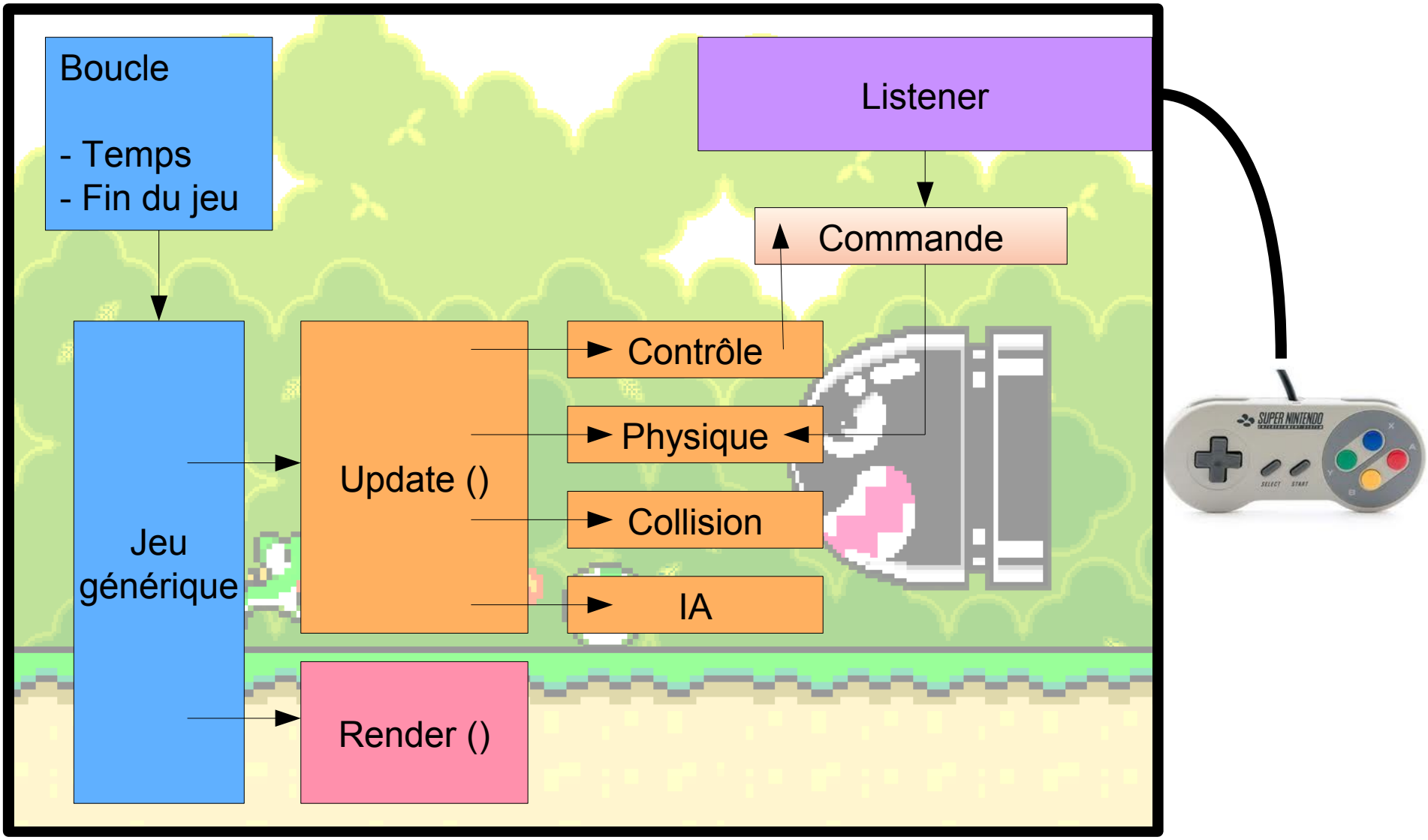

Boucle
- Temps
- Fin du jeu
Jeu générique
Update ()
Physique
Collision
IA
Render ()


SUPER NINTENDO

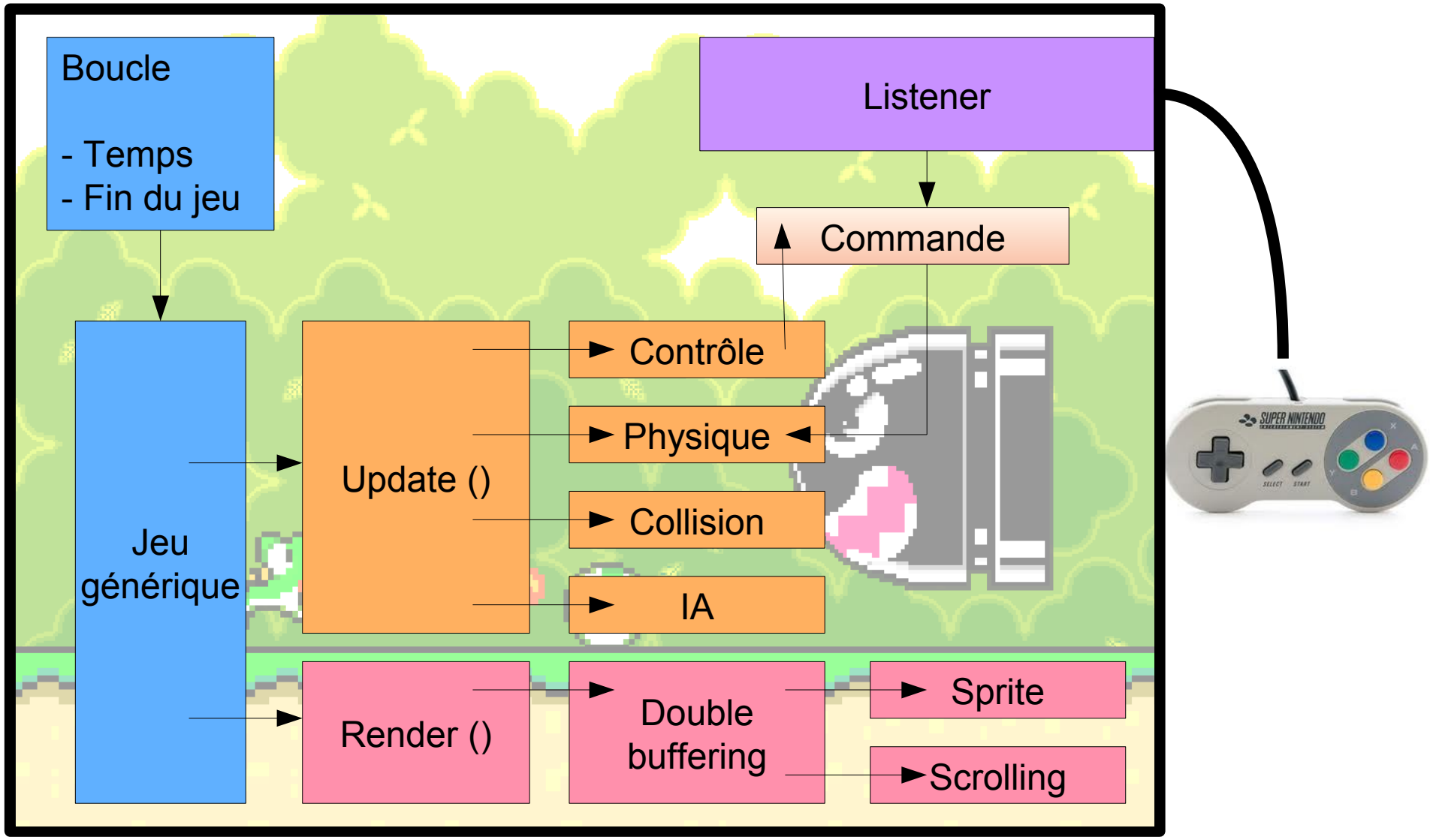
Bilan


Boucle
- Temps
- Fin du jeu
Listener
Jeu générique
Update ()
Physique
Collision
IA
Render ()
SUPER NINTENDO

Bilan


Boucle
- Temps
- Fin du jeu
Listener
Commande
Jeu
générique
Update ()
Contrôle
Physique
Collision
IA
Render ()
SUPER NINTENDO

Bilan


Boucle
- Temps
- Fin du jeu
Listener
Commande
Jeu générique
Update ()
Contrôle
Physique
Collision
IA
Render ()
Double buffering
Sprite
Scrolling

Quelques conseils

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

Quelques ressources

- 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