# Puissance 4
# Construire un projet

Christophe Viroulaud

Première - NSI

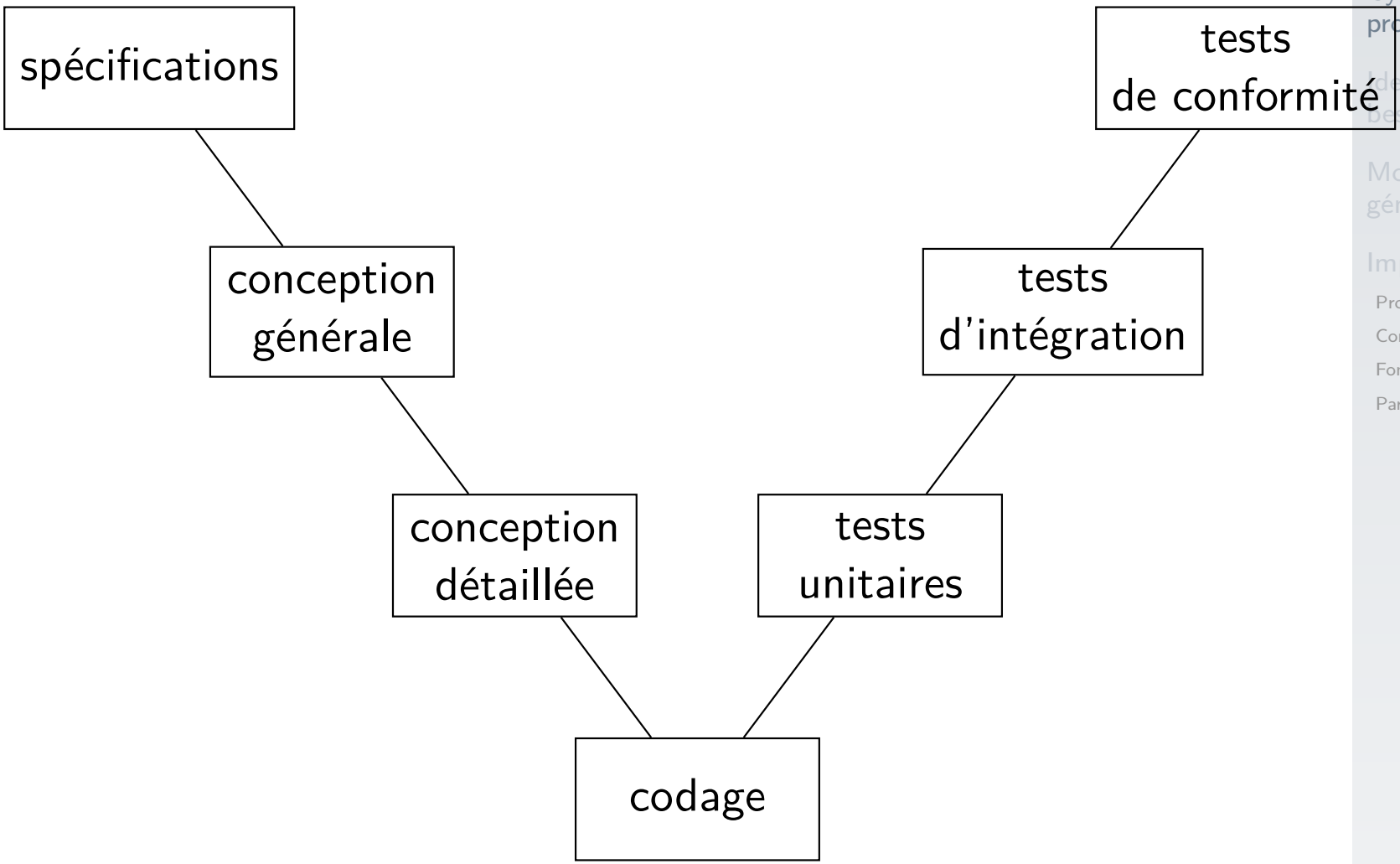
**Lang 10**

FIGURE 1 – Le *Puissance 4* est un jeu de stratégie en duel.

Comment construire un projet ?

# Sommaire

# Cycle de vie d'un projet



## Observation

Une grande partie du projet peut se réaliser sans machine.

# Sommaire

# Identification des besoins

**Spécifications :**
règles du jeu

## À retenir

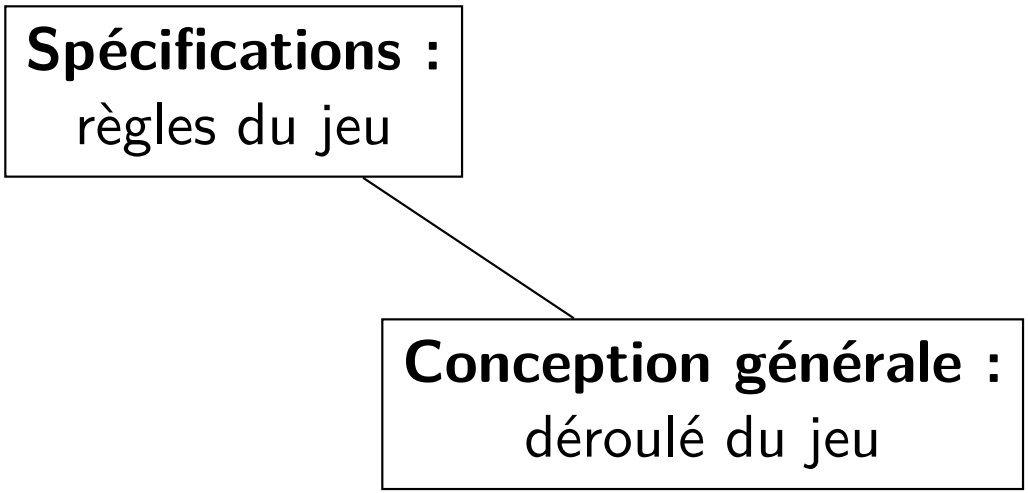
Généralement le **cahier des charges** définit les contours du projet.

Règles du puissance 4 :

- une grille de 7 colonnes et 6 lignes,
- 2 joueurs en alternance (rouge et jaune),
- gagnant : 4 pions horizontaux ou verticaux.

# Sommaire

# Modélisation générale




## À retenir

Il s'agit de définir un **algorithme général** du jeu.

**Activité 1 :** Écrire un algorithme qui décrit le déroulement d'une partie.

# Correction

Initialiser une grille vide.
Tant qu'il n'y a pas de gagnant :

- Définir le joueur en cours.
- Demander la colonne choisie et vérifier qu'elle est libre.
- Placer le jeton dans la colonne.
- Afficher la grille.
- Vérifier si le placement est gagnant :
  - si oui : partie terminée,
  - si non : recommencer un tour.

Afficher le gagnant.

**Remarque**

Cet algorithme n'est pas unique.

# Conception détaillée

**Spécifications :**
règles du jeu

**Conception générale :**
déroulé du jeu

**Conception détaillée :**
découpage en fonctions
signatures des fonctions

## À retenir

Il s'agit de détailler chaque étape de l'algorithme général. Dans un programme on confie chaque tâche à **une fonction**.

**Initialiser** une grille vide.
Tant qu'il n'y a pas de gagnant :

- **Définir** le joueur en cours.
- **Demander la colonne** choisie et **vérifier** qu'elle est libre.
- **Placer le jeton** dans la colonne.
- **Afficher** la grille.
- **Vérifier si le placement** est gagnant :
  - si oui : partie terminée,
  - si non : recommencer un tour.

**Afficher** le gagnant.

## `initialiser_grille`

- rôle : construire la grille du jeu. Une place vide est marquée par un zéro.
- paramètres :
  - col : entier
  - lig : entier
- renvoie un tableau de **lig** tableaux contenant chacun **col** zéros.

**Activité 2 :** Déterminer une signature possible des fonctions suivantes :

- `verif_gagnant`
- `choisir_colonne`
- `est_remplie`

# Correction

## `verif_gagnant`

- rôle : vérifie si le dernier jeton placé crée un alignement
- paramètres :
  - grille : tableau
  - joueur : entier, joueur en cours
  - ligne : entier, ordonnée du dernier jeton
  - colonne : entier, abscisse du dernier jeton
- renvoie un booléen : vrai si le joueur a gagné

## `choisir_colonne`

- rôle : demande la colonne où poser le jeton
- paramètres : aucun
- renvoie un entier : la colonne choisie

## `est_remplie`

- rôle : vérifie si la colonne est remplie jusqu'en haut.
- paramètres :
  - grille : tableau
  - colonne : entier
- renvoie un booléen : vrai si la colonne est remplie.

# Correction

## `placer_jeton`

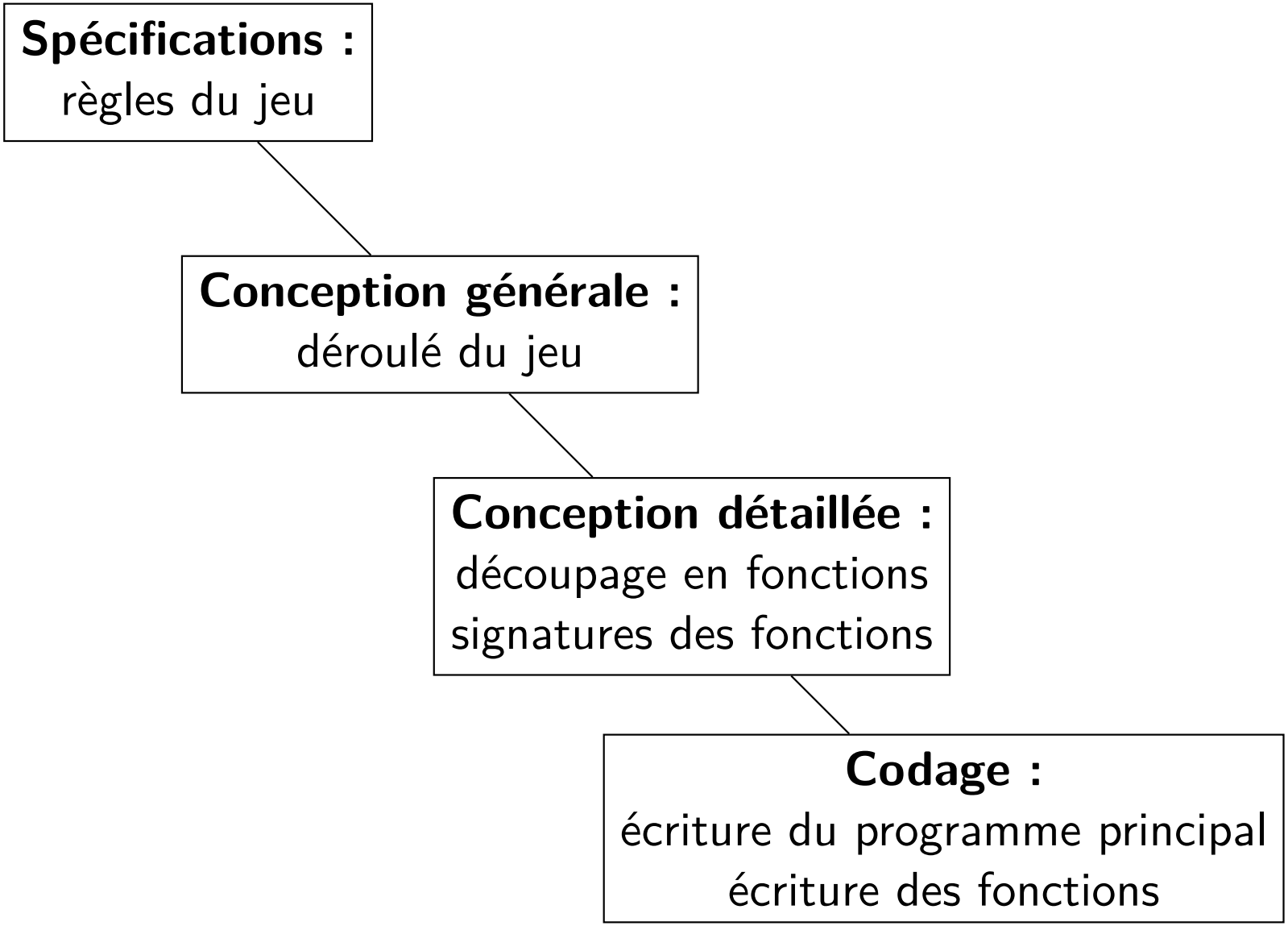
- rôle : place le jeton dans la colonne choisie en le faisant descendre dans la ligne la plus basse possible
- paramètres :
  - grille : tableau
  - colonne : entier
  - joueur : entier
- renvoie un entier : la ligne où le jeton est placé

## Remarque

Ces signatures ne sont pas uniques et dépendent des réflexions de l'équipe. Cependant, une fois l'architecture fixée, **chaque développeur doit la respecter exactement**.

# Sommaire

# Implémenter - Programme principal



## À retenir

Implémenter le programme consiste à **transformer en code informatique** l'algorithme modélisé.
Cette étape se déroule (enfin) sur la machine.

**Activité 3 :**

1. Télécharger et extraire le dossier compressé **`puissance4-annexe.zip`**
2. Ouvrir le fichier **`puissance4_console.py`**
3. Repérer les étapes de l'algorithme général.

**Initialiser** une grille vide.
Tant qu'il n'y a pas de gagnant :

- **Définir** le joueur en cours.
- **Demander la colonne** choisie et **vérifier** qu'elle est libre.
- **Placer le jeton** dans la colonne.
- **Afficher** la grille.
- **Vérifier si le placement** est gagnant :
  - si oui : partie terminée,
  - si non : recommencer un tour.

**Afficher** le gagnant.

```
1 grille = initialiser_grille(LARGEUR, HAUTEUR)
```

Code 1 – Initialisation

```
1 joueur = changer_joueur(joueur)
```

Code 2 – Définir le joueur en cours.

## Remarque

La variable `joueur` a été initialisée avant la boucle.

```
1 colonne = choisir_colonne()
2 while est_remplie(grille, colonne):
3     colonne = choisir_colonne()
```

Code 3 – Demander la colonne et vérifier

```
1 ligne = placer_jeton(grille, colonne, joueur)
```

Code 4 – Placer le jeton

**Remarque**

On récupère la valeur de la `ligne`.

```
1 while not verif_gagnant(grille, joueur, ligne, colonne):
```

Code 5 – Vérifier le gagnant pour effectuer ou non un tour supplémentaire.

## Remarques

- Le programme principal utilise les fonctions **en respectant les signatures définies dans la conception**.

## Remarques

- Le programme principal utilise les fonctions **en respectant les signatures définies dans la conception**.
- Les fonctions sont implémentées dans d'autres fichiers puis importées dans le programme principal.

# Sommaire

# Constantes

- Certaines caractéristiques du jeu sont définies une fois pour toute dans le fichier `constantes.py`
- Ces constantes sont ensuite utilisées dans tous les fichiers.

# Sommaire

# Fonctions

**Activité 4 :**

1. Ouvrir le fichier `fonctions_placement.py`
2. Compléter la fonction `initialiser_grille` en construisant la grille par compréhension.
3. Compléter la fonction `est_remplie` qui vérifie si la colonne est remplie.

# Correction



```python
def initialiser_grille(col: int, lig: int) -> list:
    """
    construire la grille du jeu. Une place vide est marquée par un zéro.

    Args:
        col (int): nombre de colonnes
        lig (int): nombre de lignes

    Returns:
        list: un tableau de lig tableaux contenant chacun col zéros.
    """
    return [[VIDE for i in range(col)] for j in range(lig)]
```

Code 6 – Initialiser

```python
def est_remplie(grille: list, colonne: int) -> bool:
    """
    vérifie si la colonne est remplie jusqu'en haut

    Args:
        grille (list): le jeu
        colonne (int): la colonne

    Returns:
        bool: True si la colonne est remplie
    """
    return not(grille[0][colonne] == VIDE)
```

Code 7 – Colonne remplie ?

## Observation

Il suffit de vérifier si l'emplacement le plus haut dans la colonne est vide.

**Activité 5 :** Compléter la fonction `placer_jeton` ayant pour signature :

- rôle : place le jeton dans la colonne choisie en le faisant descendre dans la ligne la plus basse possible
- paramètres :
  - grille : tableau
  - colonne : entier
  - joueur : entier
- renvoie un entier : la ligne où le jeton est placé

```python
def placer_jeton(grille: list, col: int, joueur: int) -> int:
    lig = 0
    # on n'est pas en bas ni sur une case remplie
    while lig+1 < HAUTEUR and grille[lig+1][col] == VIDE:
        lig = lig + 1

    # place le jeton du joueur
    grille[lig][col] = joueur
    return lig
```

Code 8 – Place le jeton

**Activité 6 :** Étude du reste du code :

1. Comment fonctionne la fonction `verif_gagnant` ?
2. Dans la fonction `verif_verticale`, quelles sont les conditions pour que la boucle `while` soit exécutée ?

# Correction



```
1 if verif_verticale(grille, joueur, ligne, colonne) or \
2    verif_horizontale(grille, joueur, ligne, colonne) or \
3    verif_diagonale_montante(grille, joueur, ligne, colonne) or \
4    verif_diagonale_descendante(grille, joueur, ligne, colonne):
```

Code 9 – Pour gagner il suffit (**or**) qu'une des conditions soient vérifiée.

## Remarque

Pour vérifier si la partie est gagnée il suffit de regarder *vers le bas* de la grille.

```
1 while ligne < HAUTEUR and \
2       grille[ligne][colonne] == joueur and \
3       compteur < 4:
```

- on ne sort pas de la grille,
- les jetons sont de la même couleur,
- on n'a pas encore 4 jetons de même couleur.

# Sommaire

# Partie graphique

Le jeu est jouable dans un terminal. Il est possible de gérer l'affichage avec une bibliothèque graphique comme **`turtle`**.

**Activité 7 :**

1. Exécuter le programme **`python_console.py`** pour jouer une partie dans la console.
2. Exécuter le programme **`python_turtle.py`** pour jouer une partie en mode graphique.
3. Comparer le code des deux programmes.

# Correction

Seuls les imports sont modifiés :

```
1 from rendu_console import *
```

Code 10 – Jouer dans la console

```
1 from rendu_turtle import *
```

Code 11 – Jouer avec la bibliothèque `turtle`

**Observation**

Les fonctions d'affichage possèdent la même signature.
Les implémentations sont différentes.