

Puissance 4 Christophe Viroulaud Lang 08

Puissance 4

Christophe Viroulaud

Première - NSI

Lang 08

1/32

Puissance 4





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

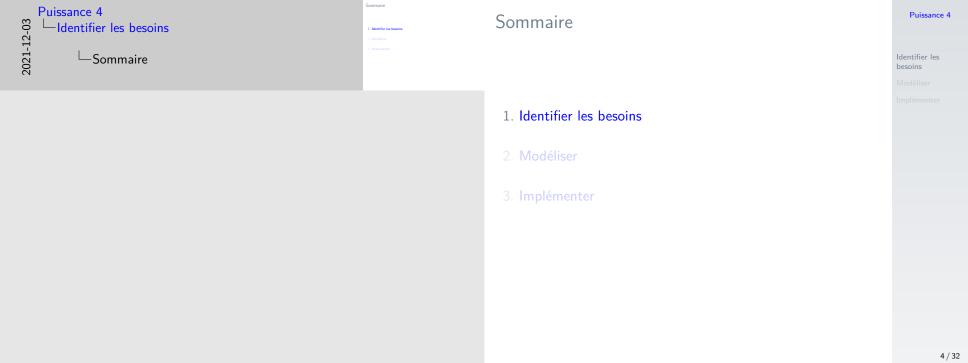
Puissance 4

entifier les esoins

1 17 .

Comment construire un projet?

Puissance 4



Identifier les besoins

└─Identifier les besoins

Puissance 4

☐ Identifier les besoins

À retenir
Il s'agit de définir les spécifications du jeu.

Identifier les besoins

Identifier les besoins

À retenir

Il s'agit de définir les **spécifications** du jeu.

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

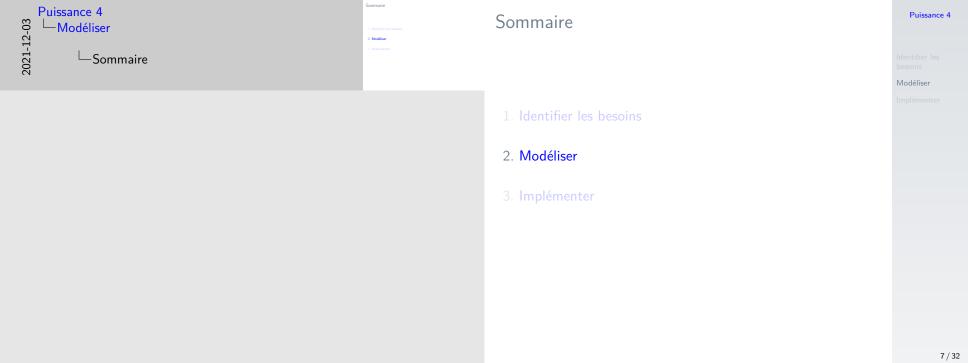
Remarque

Dans cette activité on construira une version simplifiée du jeu : on ne regardera pas les pions alignés en diagonal.

Identifier les besoins

Modéliser

mplémenter



Modéliser - conception générale

Modéliser - conception générale

A retenir

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

Activité 1 : Construire un déroulé du jeu.

Modéliser - conception générale

À retenir

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

Activité 1 : Construire un déroulé du jeu.

pesoins

Modéliser

olémenter

Demander la colonne choisie.
 Virifier que la colonne n'est pas pleine.
 Placer le jeton en le laissant tomber dans la colonne
 Virifier si le placement est gagnant:

si oui : partie terminée,
 si non : changement de joueur.

Initialiser la grille

Choisir un joueur de départ.

Tant qu'il n'y a pas de gagnant :

- ▶ Demander la colonne choisie.
- ▶ Vérifier que la colonne n'est pas pleine.
- Placer le jeton en le *laissant tomber* dans la colonne.
- ► Vérifier si le placement est gagnant :
 - si oui : partie terminée,
 - si non : changement de joueur.

Modéliser

plémenter

Modéliser

Conception détaillée

Conception détaillée

À retenir

Il s'agit de donner les signatures des fonctions nécessaires.

10/32

Vérifier que la colonne n'est pas pleine.

Vérifier si le placement est gagnant

► si oui : partie terminée. » si non : changement de joueur.

Activité 2 : Donner une signature pour chaque étape

Initialiser la grille

Choisir un joueur de départ.

Tant qu'il n'y a pas de gagnant :

- Demander la colonne choisie.
- ▶ **Vérifier** que la colonne n'est pas pleine.
- ▶ Placer le jeton en le laissant tomber dans la colonne.
- ▶ **Vérifier** si le placement est gagnant :
 - si oui : partie terminée,
 - si non : changement de joueur.

Activité 2 : Donner une signature pour chaque étape de l'algorithme.

Modéliser





12 / 32

Correction

Modéliser

Puissance 4

- initialiser_grille() -> list
- ▶ choisir_colonne() \rightarrow int
- ▶ est_remplie(grille: list, colonne: int) \rightarrow bool

Correction

bool

initialiser_grille() -> list

▶ est_remplie(grille: list, colonne: int) →

placer_jeton(grille: list, colonne: int,

ightharpoonup choisir_colonne() ightharpoonup int

joueur) ightarrow int

Puissance 4

Modéliser

bool

initialiser_grille() -> list

▶ est_remplie(grille: list, colonne: int) →

placer_jeton(grille: list, colonne: int,

verif_gagnant(grille: list, joueur: int, ligne: int, colonne: int) ightarrow bool

ightharpoonup choisir_colonne() ightharpoonup int

joueur) \rightarrow int

Correction

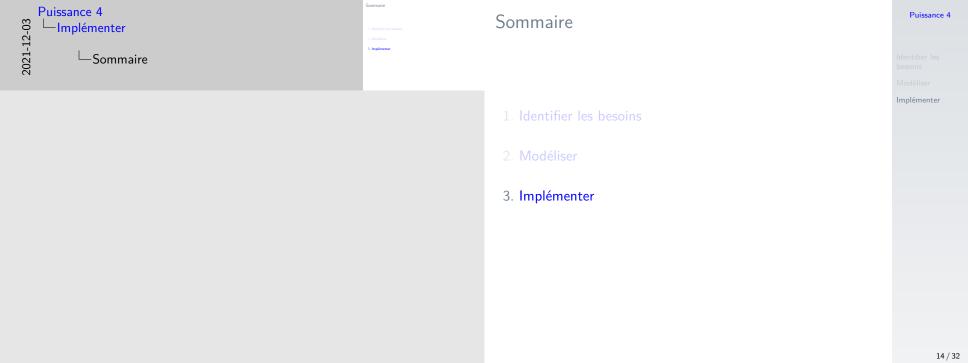
Puissance 4

Modéliser

Implémenter

Remarque

Il sera peut-être nécessaire d'écrire d'autres fonctions *internes* pour exécuter certaines tâches, rendre le code plus lisible...



Implémenter

Puissance 4

Implémenter

À retenir

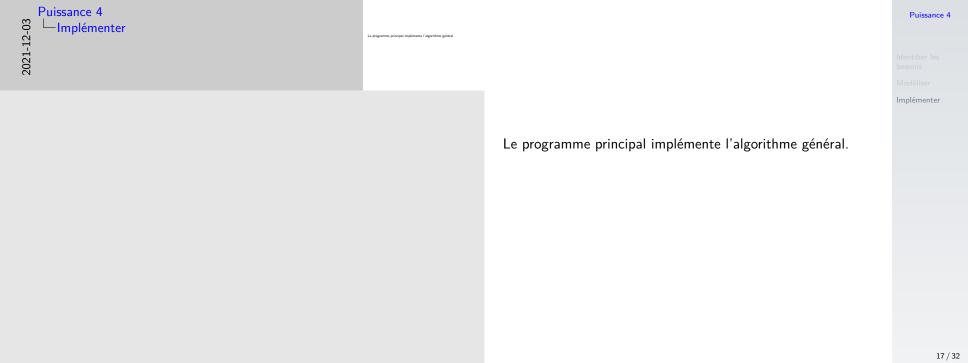
Il s'agit de transformer en code informatique l'algorithme modélisé.

Activité 3 :

- 1. Télécharger et extraire le dossier compressé puissance4-annexe.zip sur le site https://cviroulaud.github.io
- 2. Ouvrir le fichier puissance4.py

besoins

Implémenter



grille = initialiser_grille() joueur = ROUGE

Code 1 – Initialisation

Remarque

grille = initializer_grille() joueur = ROUGE Code 1 - Initialisation La couleur (ROUGE) du joueur est stockée dans une constante

La couleur (ROUGE) du joueur est stockée dans une constante.

Implémenter

```
1 respite = True
2 while respite:
3 colome = choistr_colomes()
4 colome = stronghist(gille, colomes)
Code 2 - Demander is colome et vérifer

Remarque
If faut initialiser la variable respite.
```

```
remplie = True
while remplie:
colonne = choisir_colonne()
remplie = est_remplie(grille, colonne)
```

Code 2 – Demander la colonne et vérifier

Remarque

Il faut initialiser la variable remplie.

19 / 32

Implémenter

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

Code 3 – Placer le jeton

Remarque

ligne = placer_jeton(grille, colonne, joueur)

Remarque On récupère la valeur de la ligne.

On récupère la valeur de la ligne.

dentifier les pesoins Modéliser

```
Implémenter
```

```
1 if verif_papeas(grile, jouez, lipe, colones)
2 gapast - Tros
4 class
4 class
5 jouez - charte jouez
5 jouez - charge, jouez(pouz)
Code - Vérfor le gapast
```

```
if verif_gagnant(grille, joueur, ligne, colonne):
    gagnant = True
else:
    # au tour de l'autre joueur
joueur = changer_joueur(joueur)
```

Code 4 – Vérifier le gagnant

Remarques

- Le code est découpé en fichiers puis en fonctions.
- Le programme principal est simplifié au maximum.
- ► La partie *graphique* est pour l'instant hors programme.

Implémenter

- Ouvrir le fichier constantes.py. Il contient des variables utilisables dans tout le programme. Elles ne doivent pas être modifiées.
- Ouvrir le fichier fonctions_placement.py
 Compléter la fonction initialiser_grille en
- construisant la grille par compréhension.

 4. Compléter la fonction est_remplie qui vérifie si la colonne est remplie.

Activité 4 :

- 1. Ouvrir le fichier constantes.py. Il contient des variables utilisables dans tout le programme. Elles ne doivent pas être modifiées.
- 2. Ouvrir le fichier fonctions_placement.py
- 3. Compléter la fonction **initialiser_grille** en construisant la grille par compréhension.
- 4. Compléter la fonction **est_remplie** qui vérifie si la colonne est remplie.

ldentifier les besoins

Correction

Correction

ion Puissance 4

ldentifier les besoins Modéliser

Implémenter

```
def initialiser_grille() -> list:
    """
construire la grille du jeu

Returns:
    list: un tableau de HAUTEUR lignes et
    LARGEUR colonnes

"""
return [[VIDE for i in range(LARGEUR)] for j in range(HAUTEUR)]
```

Code 5 – Initialiser

```
ind ant-emploightle list, others and -9 band of anti-
control of the control of t
```

```
def est_remplie(grille: list, colonne: int) -> bool:
       vérifie si la colonne est remplie jusqu'en haut
                                                             Implémenter
 4
 5
       Args:
            grille (list): le jeu
 6
            colonne (int): la colonne
 8
       Returns:
 9
10
            bool: True si la colonne est remplie
11
        11 11 11
        # il suffit de vérifier si l'emplacement le plus
12
       haut est vide
       return not(grille[0][colonne] == VIDE)
13
```

Code 6 - Colonne remplie?

- Activité 5 :

 1. Pour placer le jeton on écrit une fonction intermédiaire : tomber_ligne(grille: list,
- intermédiaire : tomber_ligne(grille: list, colonne: int) → int. Elle renvoie la position du jeton qui est tombé.

En utilisant la fonction précédente, compléter la fonction placer_jetos

Activité 5 :

- Pour placer le jeton on écrit une fonction intermédiaire : tomber_ligne(grille: list, colonne: int) → int. Elle renvoie la position du jeton qui est tombé.
- 2. En utilisant la fonction précédente, compléter la fonction placer_jeton

Identifier les besoins

Implémenter

```
def tomber_ligne(grille: list, colonne: int) -> int:
    ligne = 0
    while ligne < HAUTEUR and grille[ligne][colonne]
    == VIDE:
        # on descend tant qu'on n'est pas en bas ou
    sur une case remplie
        ligne = ligne + 1

# renvoie la dernière place vide
    return ligne-1</pre>
```

Code 7 – Trouve la ligne d'arrivée

def tomber_ligne(grille: list, colonne: int) -> int: ligne = 0 while ligne < HAUTEUR and grille[ligne][colonne] == VIDE:

Code 7 - Trouve la liene d'arrivée

```
1 def placer_jeton(grille: list, colonne: int, joueur) -> int:
      ligne = tomber_ligne(grille, colonne)
      grille[ligne][colonne] = joueur
      return ligne
```

Code 8 – Place le jeton

def placer_jeton(grille: list, colonne: int, joueur) -> int: ligne = tomber_ligne(grille, colonne) grille[ligne][colonne] = joueur return ligne

- Comment fonctionne la fonction verif_gagnant?
 Dans la fonction verif_verticale, quelles sont les conditions pour que la boucle while soit
- 3. Que faut-il ajouter pour vérifier les diagonales?

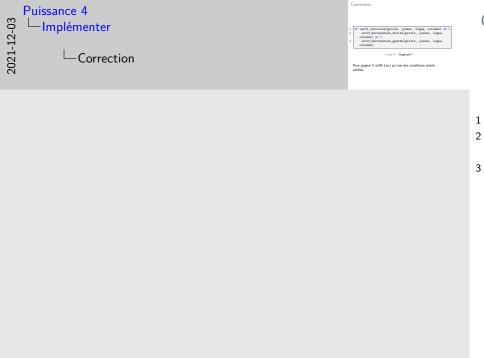
Activité 6 : Étude du reste du code :

- 1. Comment fonctionne la fonction verif_gagnant?
- 2. Dans la fonction verif_verticale, quelles sont les conditions pour que la boucle while soit exécutée?
- 3. Que faut-il ajouter pour vérifier les diagonales?

dentifier les

iviodeliser

Implémenter



Correction

Puissance 4

Implémenter

```
if verif_verticale(grille, joueur, ligne, colonne) or \
    verif_horizontale_droite(grille, joueur, ligne,
  colonne) or \
    verif_horizontale_gauche(grille, joueur, ligne,
  colonne):
```

Code 9 - Gagnant?

Pour gagner il suffit (or) qu'une des conditions soient vérifiée.

Correction

Remarque

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

```
while ligne < HAUTEUR and
       grille[ligne][colonne] == joueur and
        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.

Implémenter

Pour améliorer le jeu il faut :

- créer les fonctions verif_diagonale,
- modifier la condition de la fonction verif_gagnant.

Implémenter

```
if verif_verticale(grille, joueur, ligne, colonne) or \
verif_horizontale_droite(grille, joueur, ligne, colonne) or \
verif_horizontale_gauche(grille, joueur, ligne, colonne) or \
verif_diagonale(grille, joueur, ligne, colonne):
```