Puissance 4

mnlémenter

puissance4-annexe.zip

Puissance 4

Christophe Viroulaud

Première - NSI

DonRep 14

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

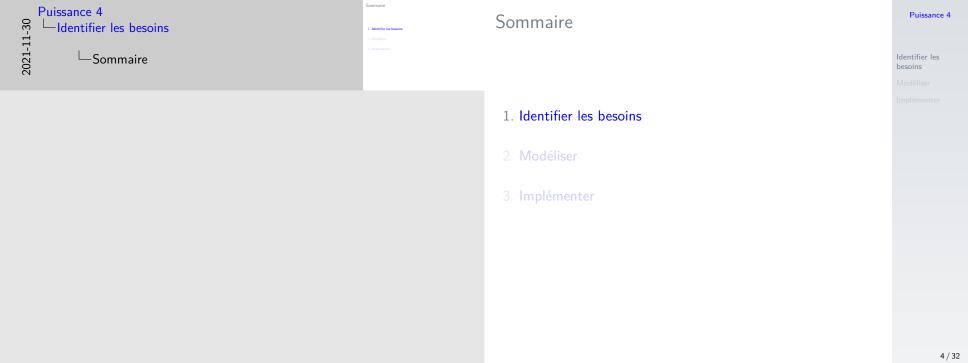
Puissance 4

entifier les esoins

Comment construire un projet ?

Comment construire un projet?

Puissance 4



Identifier les besoins

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

Puissance 4

-Identifier les besoins

☐ Identifier les besoins

r les desoins

besoins Modéliser

nplémenter

À retenir

Il s'agit de définir les ${\bf sp\'ecifications}$ du jeu.

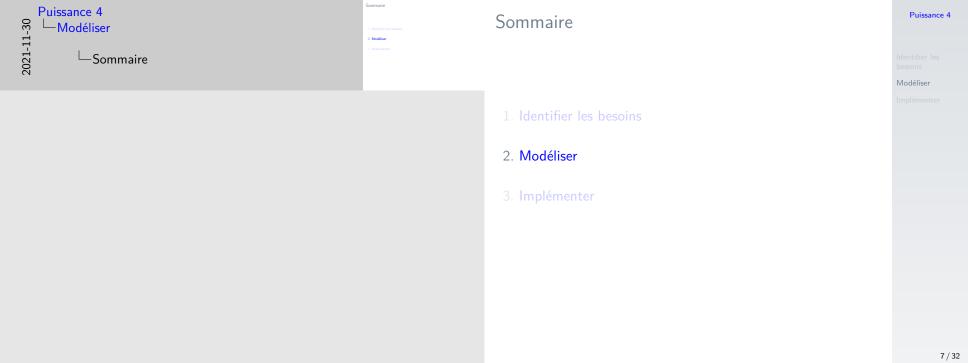
- ▶ une grille de 7 colonnes et 6 lignes,
- ▶ 2 joueurs en alternance (rouge et jaune),
- ▶ gagnant : 4 pions horizontaux ou verticaux.

Remarque

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

Identifier les besoins

Modéliser



└─Modéliser - conception générale

Modéliser - conception générale

A retenir

II v aps de définir un algorithme général du jeu.

Activité 1 : Constrain un déreolé du jeu.

Modéliser - conception générale

À retenir

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

 $\textbf{Activité 1:} \ \mathsf{Construire} \ \mathsf{un} \ \mathsf{d\'eroul\'e} \ \mathsf{du} \ \mathsf{jeu}.$

pesoins

Modéliser

Correction Choisir un joueur de départ.

Initialiser la grille

Tant qu'il n'y a pas de gagnant ▶ Demander la colonne choisie. ► Vérifier que la colonne n'est pas pleine. ► Placer le ieton en le /aissant tomber dans la colonne ► Vérifier 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

Conception détaillée

Conception détaillée

À retenir

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

- Demander la colonne choisie.

 Vérifier que la colonne n'est pas pleine.
- Placer le jeton en le Jaissant 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.

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.

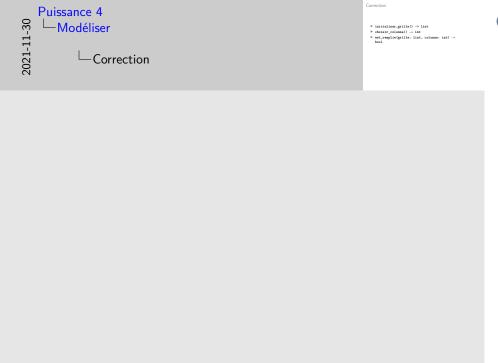
Modéliser

nplémenter





12 / 32



bool

Correction

initialiser_grille() -> list

▶ est_remplie(grille: list, colonne: int) \rightarrow

▶ choisir_colonne() \rightarrow int

Modéliser

Puissance 4

Correction

bool

ightharpoonup choisir_colonne() ightharpoonup int

joueur) ightarrow int

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

placer_jeton(grille: list, colonne: int,

Puissance 4

Modéliser

initialiser_grille() -> list

Correction

Puissance 4

Modéliser

initialiser_grille() -> list

ightharpoonup choisir_colonne() ightharpoonup int

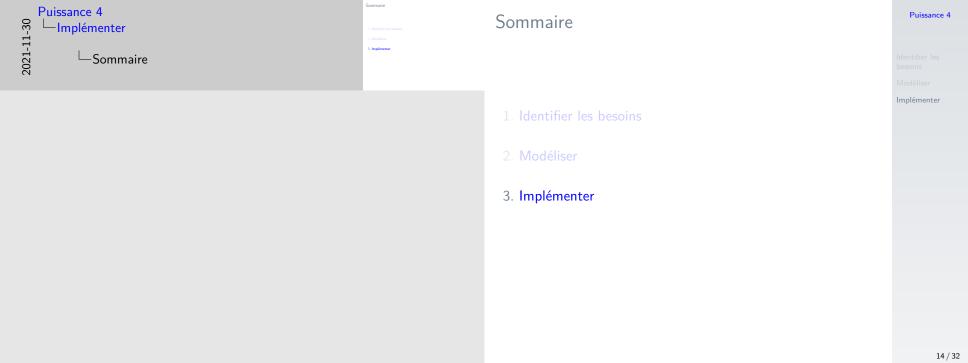
▶ est_remplie(grille: list, colonne: int) → bool

placer_jeton(grille: list, colonne: int, joueur) \rightarrow int

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

Remarque

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



Implémenter

Puissance 4

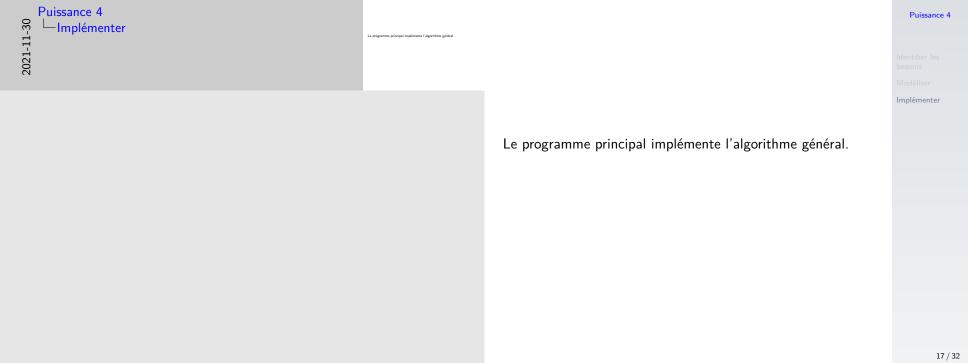
Implémenter

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



Puissance 4

entifier les soins

Implémenter

```
grille = initialiser_grille()
joueur = ROUGE
```

 $\mathsf{Code}\ 1 - \mathsf{Initialisation}$

Remarque

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

Implémenter

```
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

Puissance 4

Identifier les besoins

Implémenter

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

Code 3 – Placer le jeton

Remarque

On récupère la valeur de la ligne.

dentifier les pesoins Modéliser

```
i [if wert_papawi(griis, ]ower, lign, colomo);
2 papawi *Tree
2 papawi *Tree
3 papawi *Tree
4 papawi *Tree
5 power *changer_power(power)

Colo 4 - Verfer is papawi
```

```
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.

Ouvrir le fichier fonctions_placement.py
 Compléter la fonction initializer_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.

Modéliser

Correction

Correction

ction

besoins Modéliser

Puissance 4

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

```
of ent_replicignile list, others int) -9 built

writes it a minma and respite jump'un hand

degregation (mark) in jum

description (mark) in jum

description (mark) in sinue

Markette

multi-result in columns and respite

at it market and vertices and replicament in plan

extense satisfull(Dillandman)

Conto Colors Columns and Colors

Conto Columns and Colors

Conto Colors Columns and Colors

Conto Colors Columns and Colors

Colors Colors Columns and Colors

Colors Colors Columns and Colors

Colors Colors Colors

Colors Colors

Colors Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Colors

Colors Co
```

```
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?

fonction placer_jeton

Activité 5:

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

Correction

Implémenter

Puissance 4

```
def tomber_ligne(grille: list, colonne: int) -> int:
      ligne = 0
2
      while ligne < HAUTEUR and grille[ligne][colonne]</pre>
     == VIDE:
           # on descend tant qu'on n'est pas en bas ou
4
     sur une case remplie
          ligne = ligne + 1
6
      # renvoie la dernière place vide
      return ligne-1
8
```

Code 7 – Trouve la ligne 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

Activité 6 : Étude du reste du code

- 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?

uissance 4

ldentifier les besoins



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:</pre>
```

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

dentifier les esoins

1 if verif_verticale(grille, joueur, ligne, colonne) or verif_horizontale_dreste(grille, joueur, ligne, colonne) or \
1 verif_horizontale_grache(grille, joueur, ligne, colonne) or \
1 verif_delegonale(grille, joueur, ligne, colonne):

Pour améliorer le jeu il faut :

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

```
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):
```