puissance4-annexe.zip

Puissance 4

Christophe Viroulaud

Première - NSI

DonRep 13

Puissance 4

Christophe Viroulaud

Première - NSI

DonRep 13

Puissance 4

Identifier les besoins

Modéliser

Implémenter

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

Puissance 4

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

Identifier les besoins
Modéliser
Implémenter

Comment construire un projet ?

Identifier les besoins
Modéliser
Implémenter

Comment construire un projet ?

Sommaire

1. Identifier les besoins
2. Modéliser
3. Implémenter

Sommaire

1. Identifier les besoins
2. Modéliser
3. Implémenter

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

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.

- 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.

Sommaire

Sommaire

1. Identifier les besoins
2. Modéliser
3. Implémenter

Modéliser - conception générale

À 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.

Correction

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.

Correction

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.

Puissance 4
Identifier les besoins
Modéliser
Implémenter

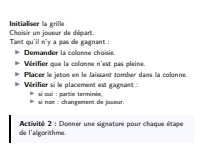
Conception détaillée

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

Conception détaillée

À retenir

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


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.

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.

Correction

- initialiser_grille() -> list

Correction

- `initialiser_grille() -> list`

Puissance 4

Identifier les besoins

Modéliser

Implémenter

Correction

- initialiser_grille() -> list
- choisir_colonne() → int

Correction

- `initialiser_grille() -> list`
- `choisir_colonne()` → `int`

Correction

- initialiser_grille() -> list
- choisir_colonne() → int
- est_remplie(grille: list, colonne: int) → bool

Correction

- `initialiser_grille() -> list`
- `choisir_colonne()` → `int`
- `est_remplie(grille: list, colonne: int)` → `bool`

Correction

- initialiser_grille() -> list
- choisir_colonne() → int
- est_remplie(grille: list, colonne: int) → bool
- placer_jeton(grille: list, colonne: int, joueur) → int

Correction

- `initialiser_grille() -> list`
- `choisir_colonne() → int`
- `est_remplie(grille: list, colonne: int) → bool`
- `placer_jeton(grille: list, colonne: int, joueur) → int`

Correction

- initialiser_grille() -> list
- choisir_colonne() → int
- est_remplie(grille: list, colonne: int) → bool
- placer_jeton(grille: list, colonne: int, joueur) → int
- verif_gagnant(grille: list, joueur: int, ligne: int, colonne: int) → bool

Correction

- `initialiser_grille() -> list`
- `choisir_colonne()` → `int`
- `est_remplie(grille: list, colonne: int)` → `bool`
- `placer_jeton(grille: list, colonne: int, joueur)` → `int`
- `verif_gagnant(grille: list, joueur: int, ligne: int, colonne: int)` → `bool`

Puissance 4
Identifier les besoins
Modéliser
Implémenter

Remarque
Il sera peut-être nécessaire d'écrire d'autres fonctions pour

Remarque

Il sera peut-être nécessaire d'écrire d'autres fonctions pour

Sommaire

Sommaire

1. Identifier les besoins
2. Modéliser
3. Implémenter

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

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**

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**

Le programme principal implémente l'algorithme général.

Le programme principal implémente l'algorithme général.

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

Code 1 – Initialisation

Remarque

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


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

Code 1 – Initialisation

Remarque

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

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

Code 2 – Initialisation

Remarque

Il faut initialiser la variable **remplie**.


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

Code 2 – Initialisation

Remarque

Il faut initialiser la variable **remplie**.

Puissance 4

Identifier les besoins

Modéliser

Implémenter

1 ligne = placer_jeton(grille, colonne, joueur)
Code 3 – Initialisation
Remarque
On récupère la valeur de la ligne.


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

Code 3 – Initialisation

Remarque

On récupère la valeur de la **`ligne`**.

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

Code 4 – Initialisation

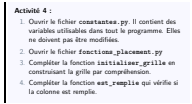

```
if verif_gagnant(grille, joueur, ligne, colonne):
    gagnant = True
else:
    # au tour de l'autre joueur
    joueur = changer_joueur(joueur)
```

Code 4 – Initialisation

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.

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.


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.

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.

Correction

Code 5 – Initialiser

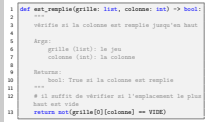
Correction

```python
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

Puissance 4
Identifier les besoins
Modéliser
Implémenter



```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
    """
    # il suffit de vérifier si l'emplacement le plus haut est vide
    return not(grille[0][colonne] == VIDE)
```

Code 6 – Colonne remplie ?

Activité 5 :
1. 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`

Activité 5 :

1. 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`**

Correction

Code 7 – Trouve la ligne d'arrivée

Correction

Puissance 4
Identifier les besoins
Modéliser
Implémenter

```python
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
```

Code 7 – Trouve la ligne d'arrivée

1 def placer_jeton(grille: list, colonne: int, joueur) -> int:
2 ligne = tomber_ligne(grille, colonne)
3 grille[ligne][colonne] = joueur
4 return ligne
Code 8 – Place le jeton


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

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 ?
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 ?

Correction

```
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 ?

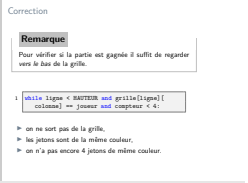
Pour gagner il suffit (or) qu'une des conditions soient vérifiée.

Correction


```
1 if verif_verticale(grille, joueur, ligne, colonne) or \
2     verif_horizontale_droite(grille, joueur, ligne,
    colonne) or \
3     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.

```
1 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.

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):
```


Pour améliorer le jeu il faut :

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

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