

## Exécution normale du jeu Pas Cman

### Exemple de lancement du jeu Pas Cman (un processus par terminal)

- **Serveur :**

```
./pas_server 9090 ./resources/map3.txt
```

- **Client 1 :**

```
./pas_client localhost 9090
```

- **Client 2 :**

```
./pas_client localhost 9090
```

# Exécution des tests via le programme `pas\_labo`

## Exemple de lancement du programme de tests

- **Labo :**

```
./pas_labo 9090 ./test/map.txt ./test/joueur1.txt ./test/joueur2.txt
```

où

9090 → port d'écoute du serveur (en *localhost*)

map.txt → carte utilisée pour le test

joueur1.txt → mouvements du joueur 1 pour le test 1

joueur2.txt → mouvements du joueur 2 pour le test 1

Les mouvements sont encodés dans des fichiers texte avec les caractères :

'v' (DOWN), '>' (RIGHT), '<' (LEFT), '^' (UP).

## Modifications à apporter à `pas\_client`

Pour que le programme de tests `pas\_labo` puisse synchroniser les mouvements des deux joueurs, nous allons apporter une petite modification au programme `pas\_client`.

- **Client lancé manuellement : 2 paramètres**

```
./pas_client localhost 9090
```

`pas\_client` lit les déplacements de son joueur sur le pipe relié à l'ui `pas\_cman\_ipl`.

- **Client lancé par `pas\_labo` : 3 paramètres**

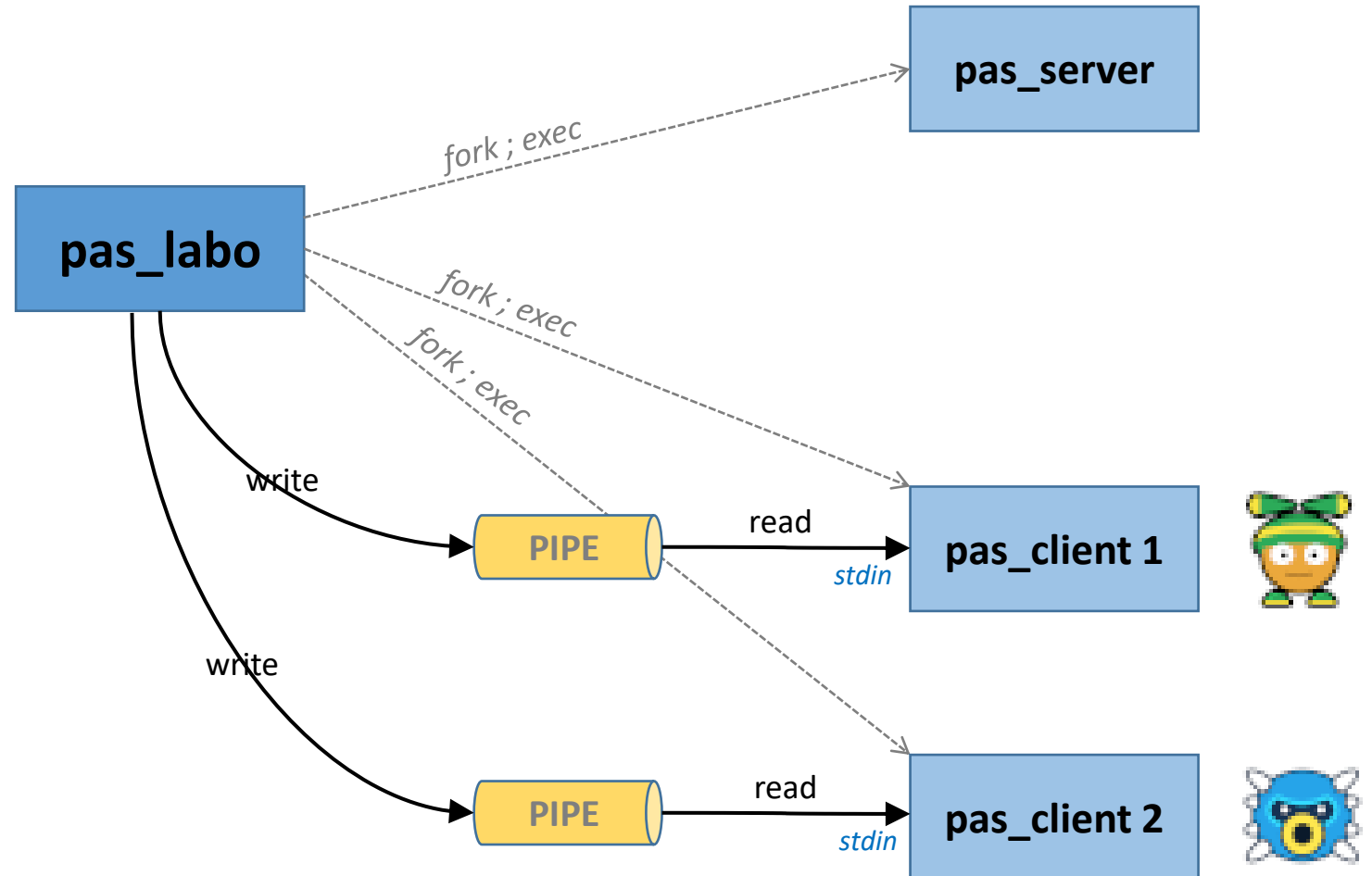
```
./pas_client localhost 9090 -test
```

Avec l'option "-test", `pas\_client` lit les déplacements de son joueur sur son entrée standard (stdin). Voir architecture dans les slides suivants.

## Architecture de `pas\_labo`

Le programme `pas\_labo` crée deux pipes et exécute 3 programmes:

- le serveur `pas\_server` en lui fournissant la map
- attend 2/10<sup>e</sup> de seconde (`usleep(200000)`)
- le 1<sup>er</sup> `pas\_client` (avec l'option « -test »)
- attend 2/10<sup>e</sup> de seconde
- le 2<sup>e</sup> `pas\_client` (avec l'option « -test »)

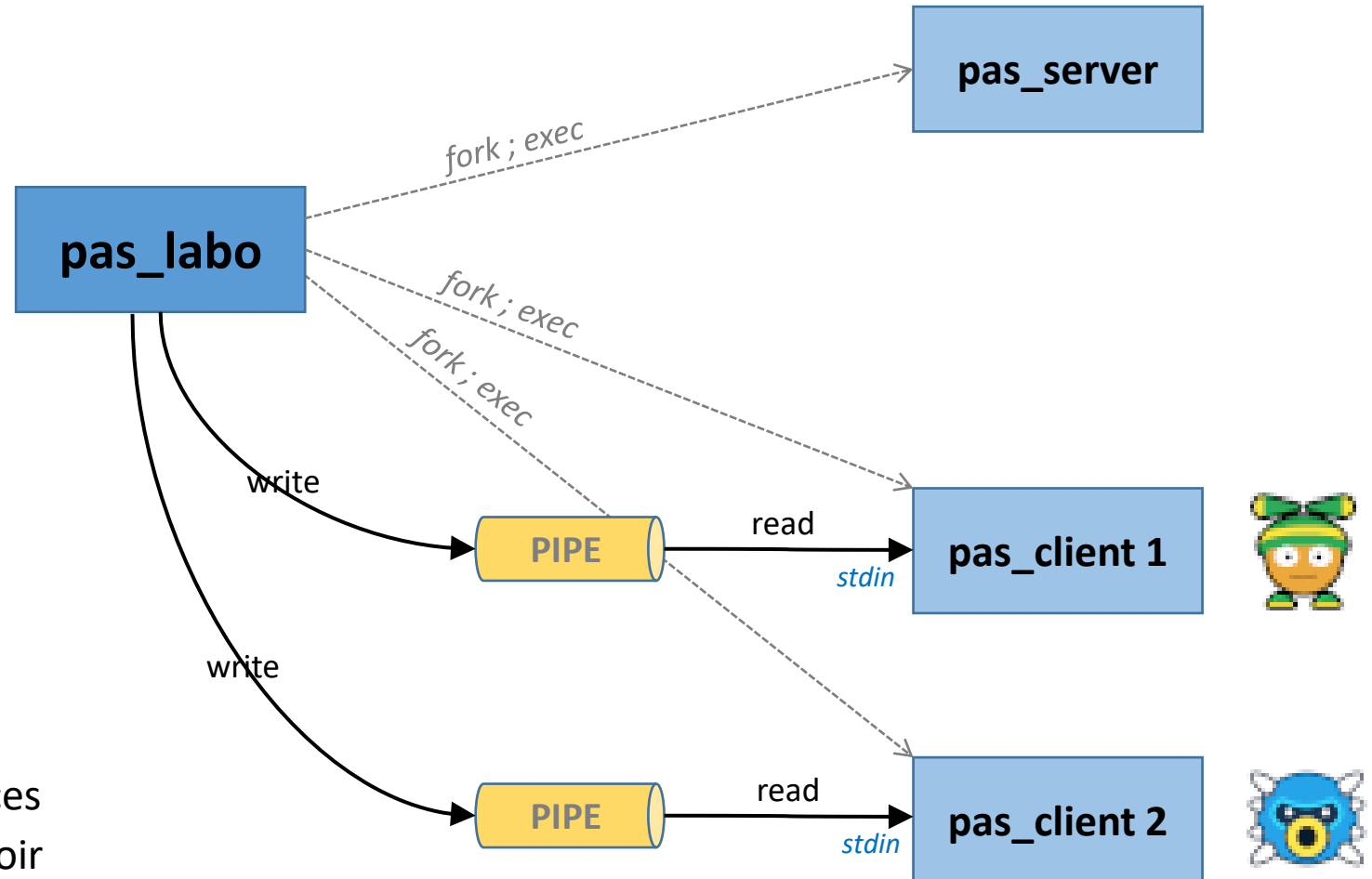


## Architecture de `pas\_labo`

Dans un boucle, `pas\_labo`:

- lit un mouvement du joueur 1 dans le fichier `joueur1.txt`
- écrit le mouvement du joueur 1 sur son pipe
- attend  $1/10^e$  de seconde (`usleep(100000)`)
- lit un mouvement du joueur 2 dans le fichier `joueur2.txt`
- écrit le mouvement du joueur 2 sur son pipe
- attend  $1/10^e$  de seconde
- et ainsi de suite jusqu'à la fin de chaque fichier.

Attend 5 secondes avant de libérer les ressources et clôturer l'exécution pour avoir le temps de voir quel joueur a gagné/perdu.

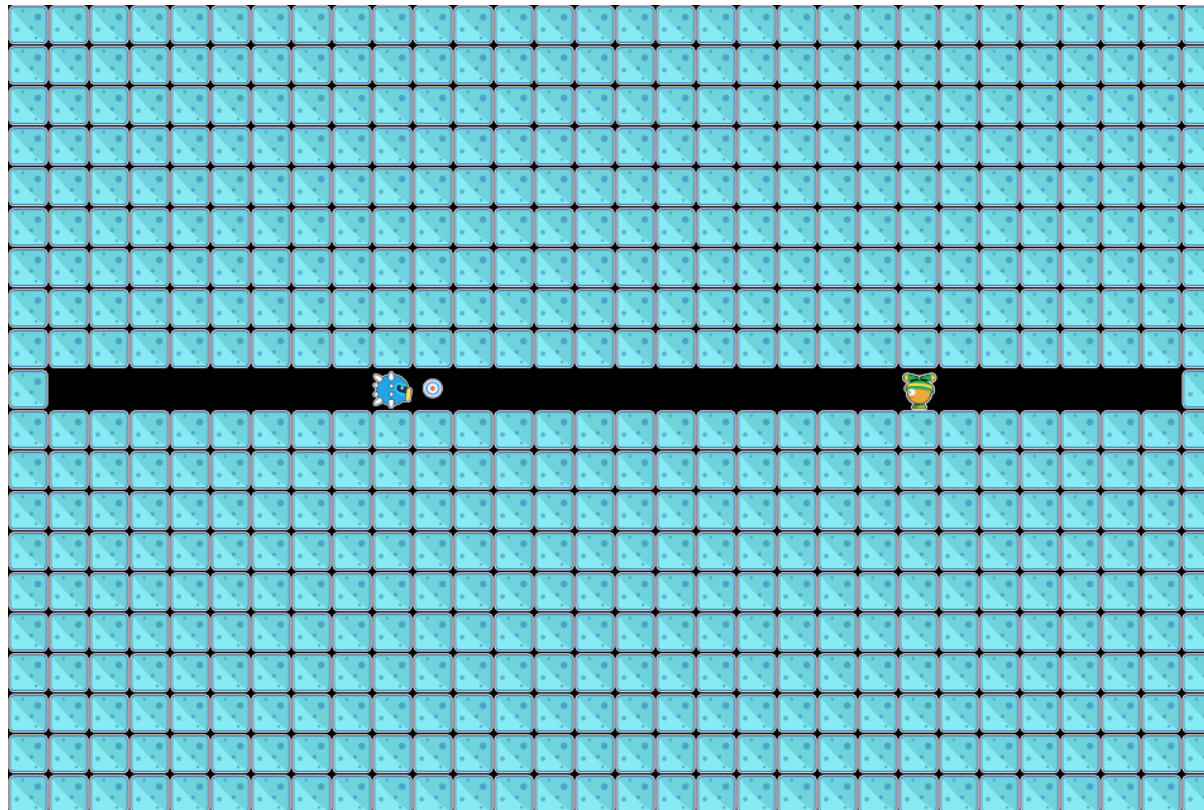


## Tests – Scénario 1

```
./pas_labo 9090 ./test1/map.txt ./test1/joueur1.txt ./test1/joueur2.txt
```

→ La partie se termine lorsque le joueur 2  mange la dernière nourriture.

→ Le joueur 1  gagne car il a mangé le canard et a donc plus de points que l'autre joueur.

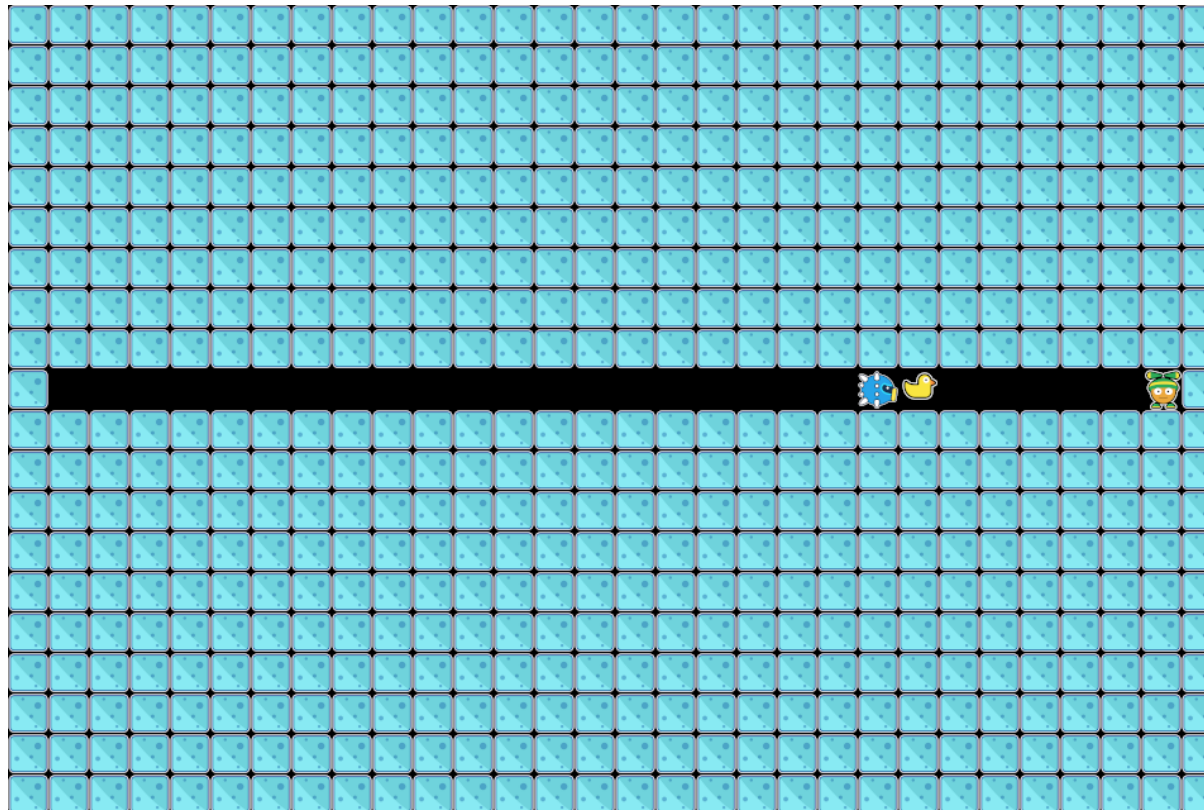


## Tests – Scénario 2

`./pas_labo 9090 ./test2/map.txt ./test2/joueur1.txt ./test2/joueur2.txt`




→ La partie se termine lorsque le joueur 2  mange la dernière nourriture, càd. la superfood.

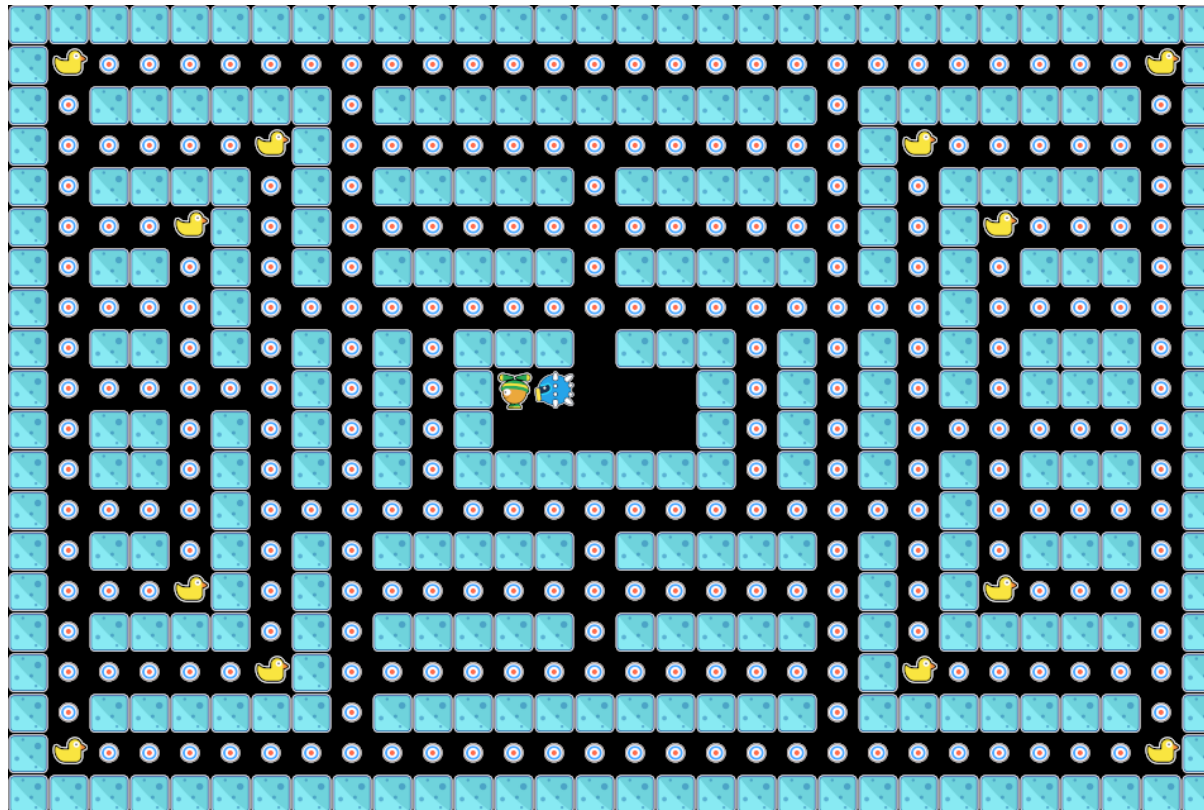
→ Le joueur 2  gagne car il a mangé toute la nourriture de la map.



## Tests – Scénario 3

```
./pas_labo 9090 ./test3/map.txt ./test3/joueur1.txt ./test3/joueur2.txt
```




- La partie se termine lorsque le joueur 2  entre en collision avec le joueur 1 .
- Le joueur 2  gagne car il est le seul a avoir mangé une nourriture.





## Tests – Scénario 4

`./pas_labo 9090 ./test4/map.txt ./test4/joueur1.txt ./test4/joueur2.txt`

- La partie se termine lorsque le joueur 2  entre en collision avec le joueur 1 .
- Le joueur 1  gagne car il a mangé plus de nourriture et superfood que l'autre joueur.

