

Projet S501

BLOCKADE



TABLE OF CONTENTS

01

**Démo et
présentation globale**

02

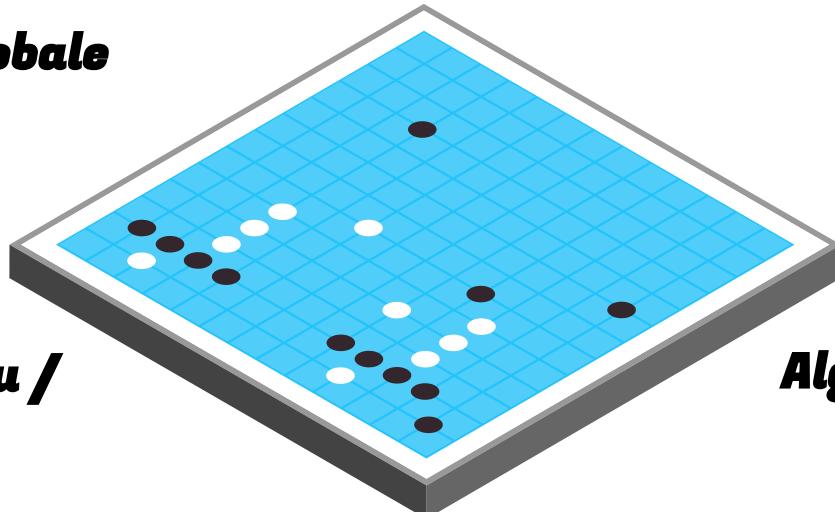
IHM

03

**Logique de Jeu /
Online**

04

Algorithme / IA



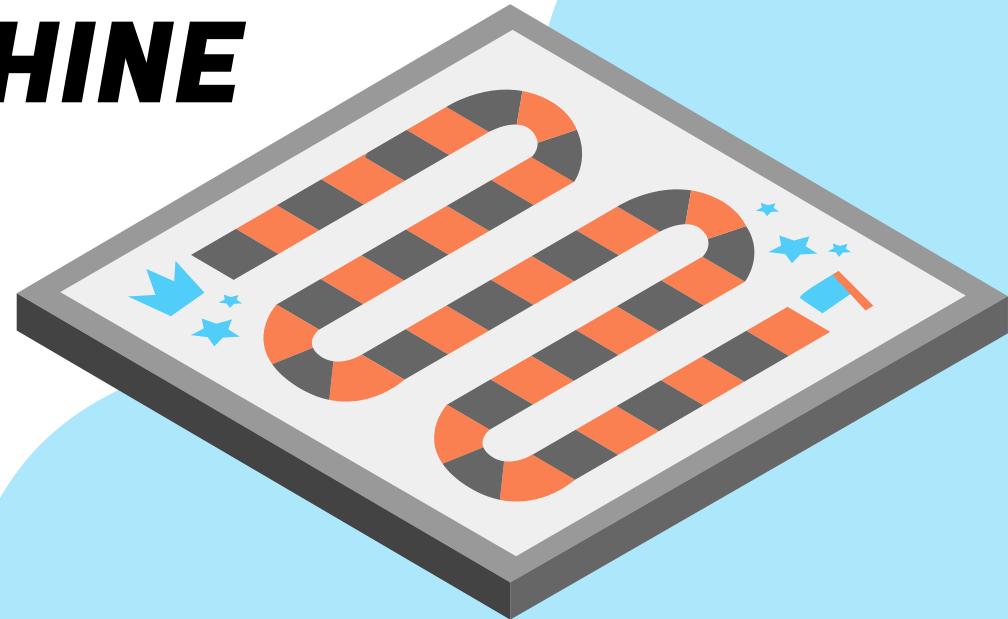
01

PRESENTATION GLOBALE



02

INTERFACE HUMAIN MACHINE (IHM)



Déroulement



Intro et DTOs

Thomas



Structure du jeu

Doha



Menus

Martin et Nolan



Animations

Wassim

Travail effectué : DTOs

A

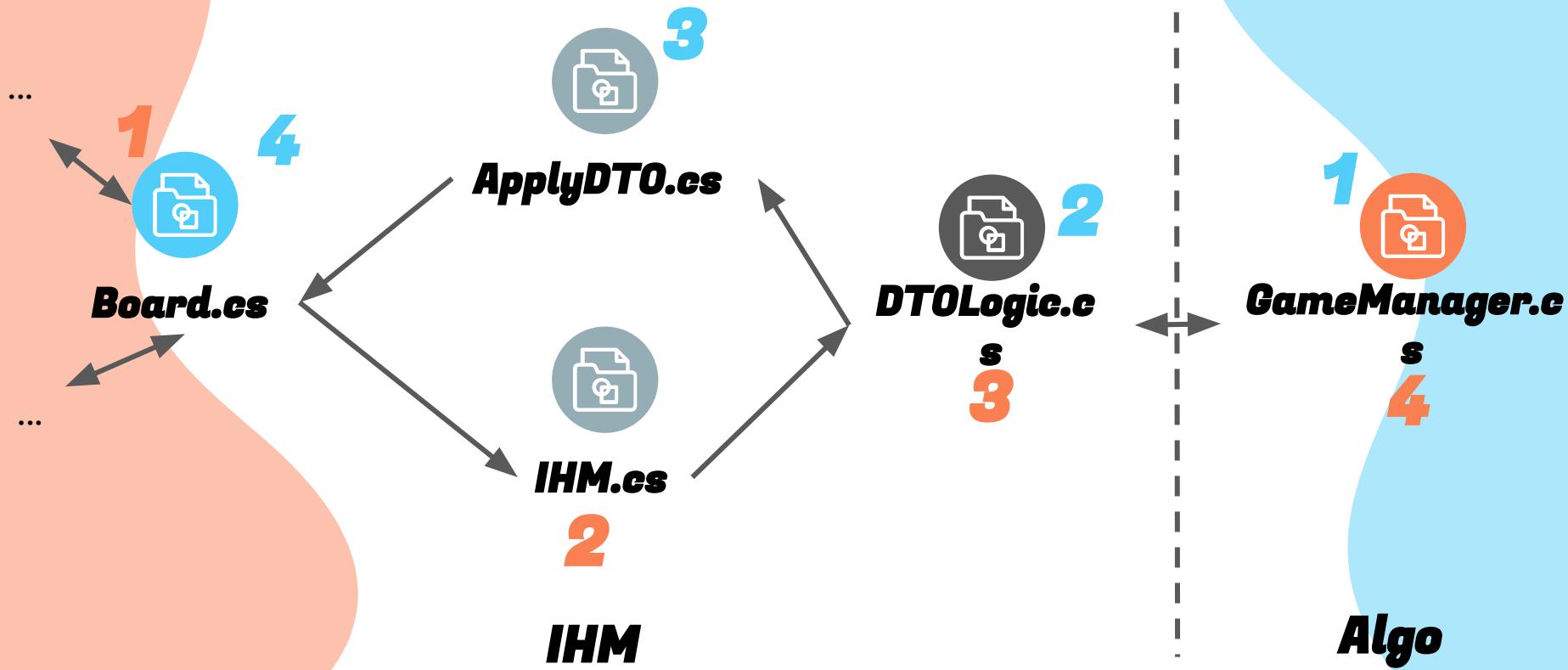
Envoi des DTOs

B

Réception des DTOs

Thomas

Travail effectué : DTOs



Travail effectué : Classe IHM



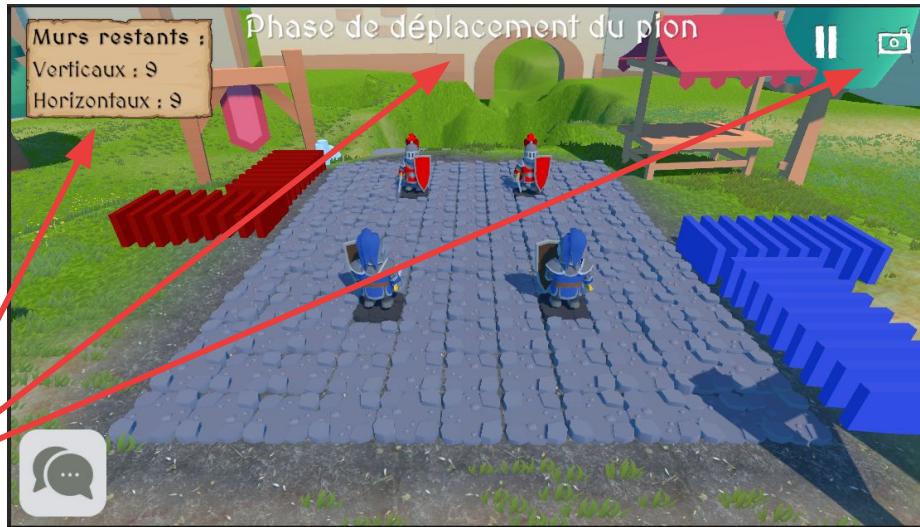
Joueurs

Structure "maison"



Overlay

Surcouche de l'interface



Déroulement d'une partie

Pion > Mur > J2 > Pion >
Mur > J1 > ...

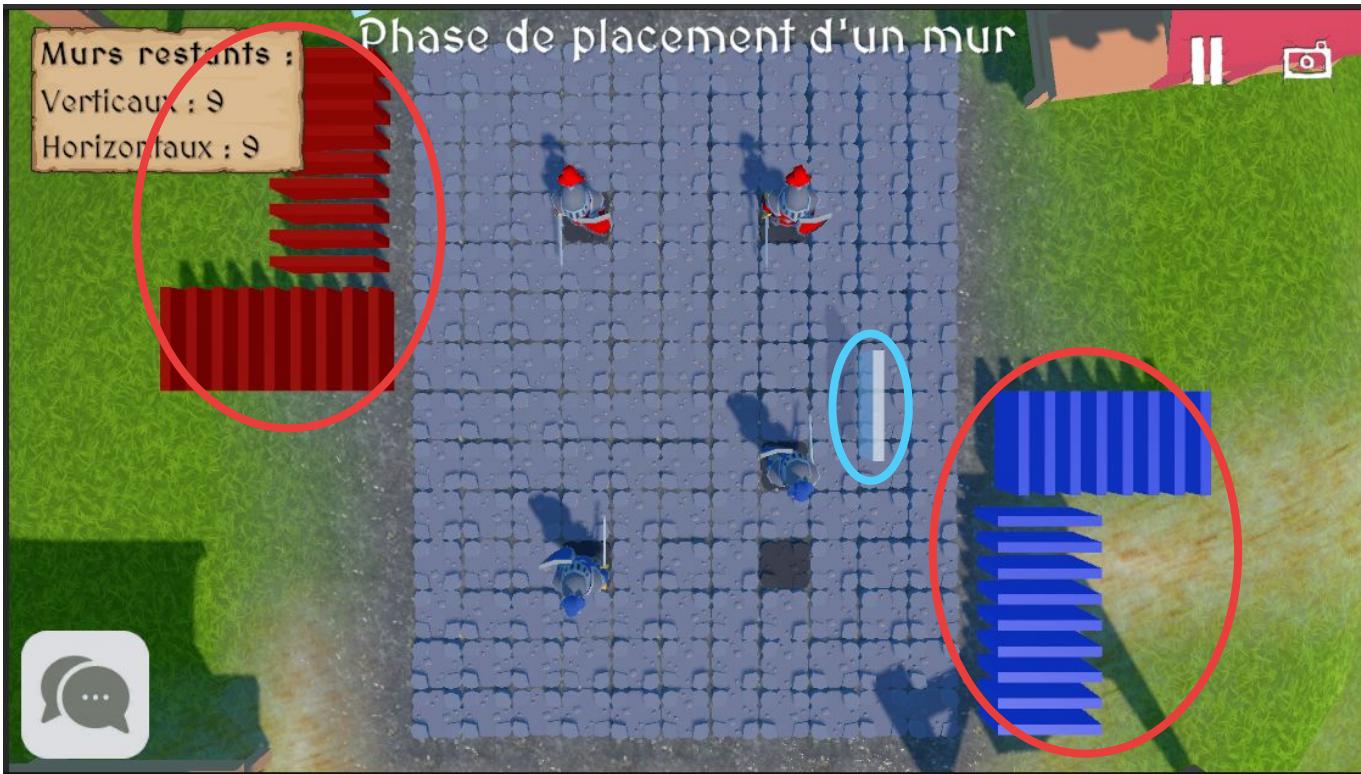


Vues & Animation

J1 / J2
Joueur / Plateau

Thomas

Travail effectué : Murs



Travail effectué : Référent

- Gestion de projet :
 - Bugfixs
 - Merges
 - Aide pour l'équipe tout au long du projet

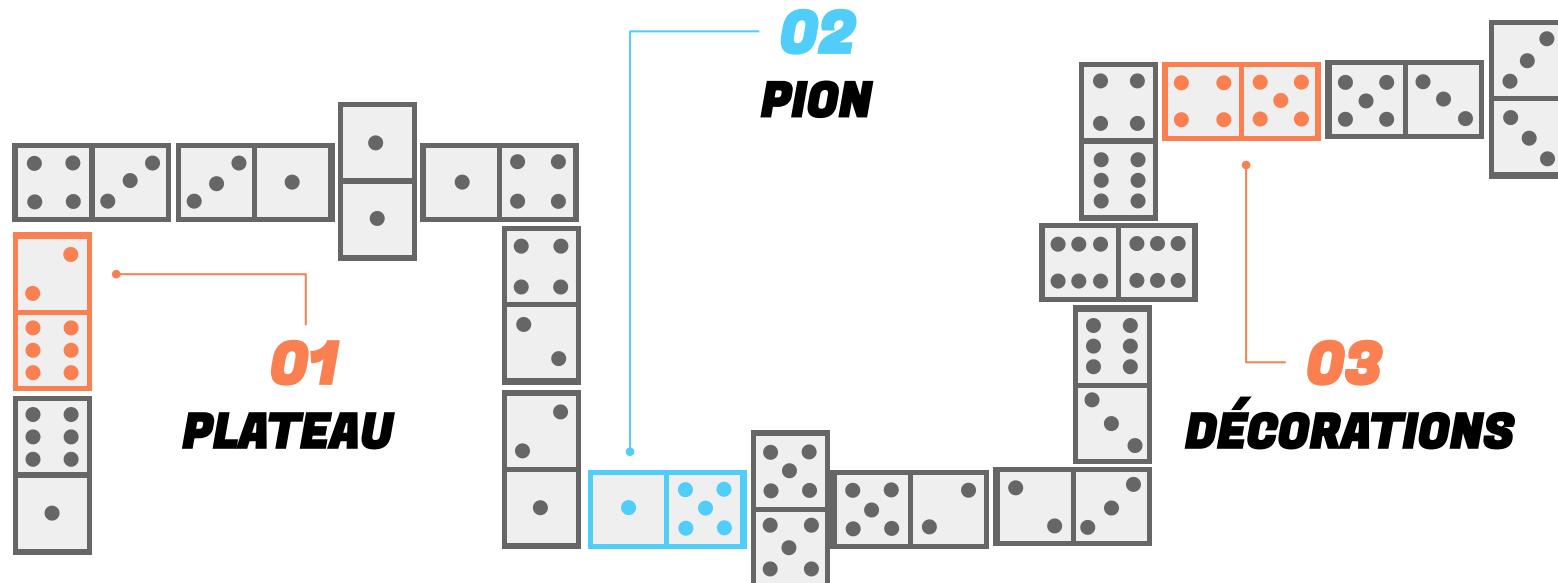


Thomas

Difficultées rencontrées



Structure du jeu



Init_Plateau()**Spécifités**

- Taille 11x14
- Instantiation des cases
- Initialisation des pions
case de départ

CaseClickHandler.cs**Pawn.cs****Effet sur les cases cliquées + son**

Pawn**Spécificités**

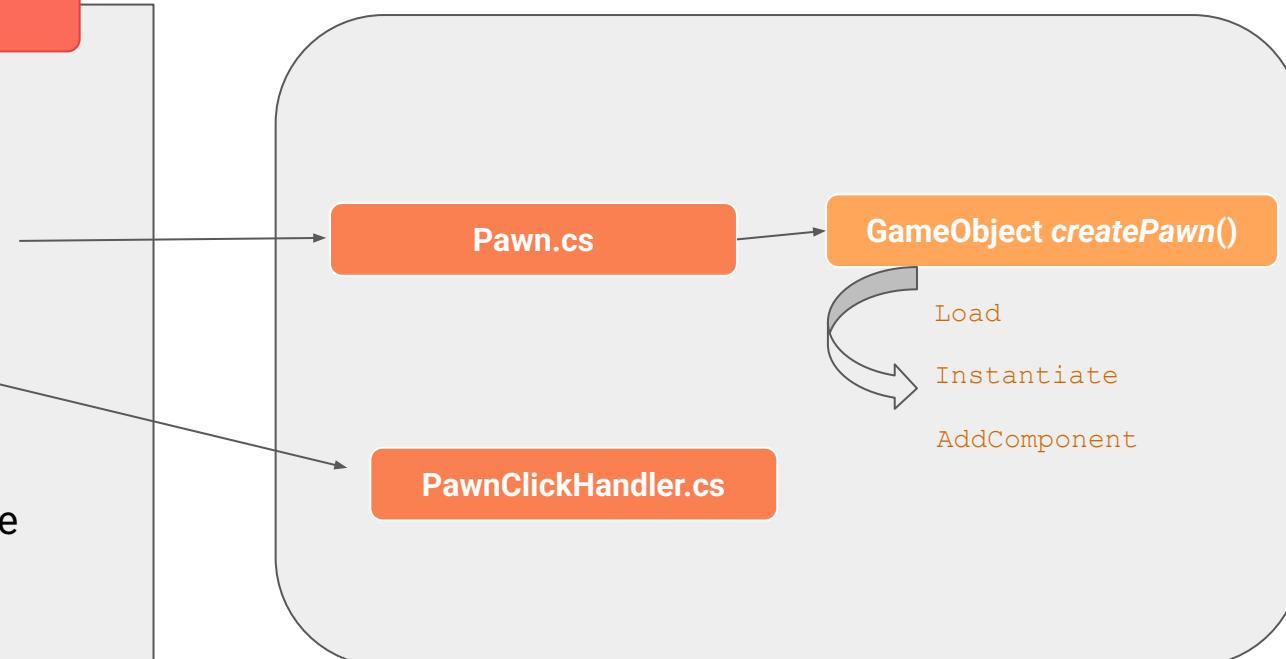
- Créer un pion
- Rendre cliquable
- Effets lorsqu'on clique

Pawn.cs**GameObject createPawn()****PawnClickHandler.cs**

Load

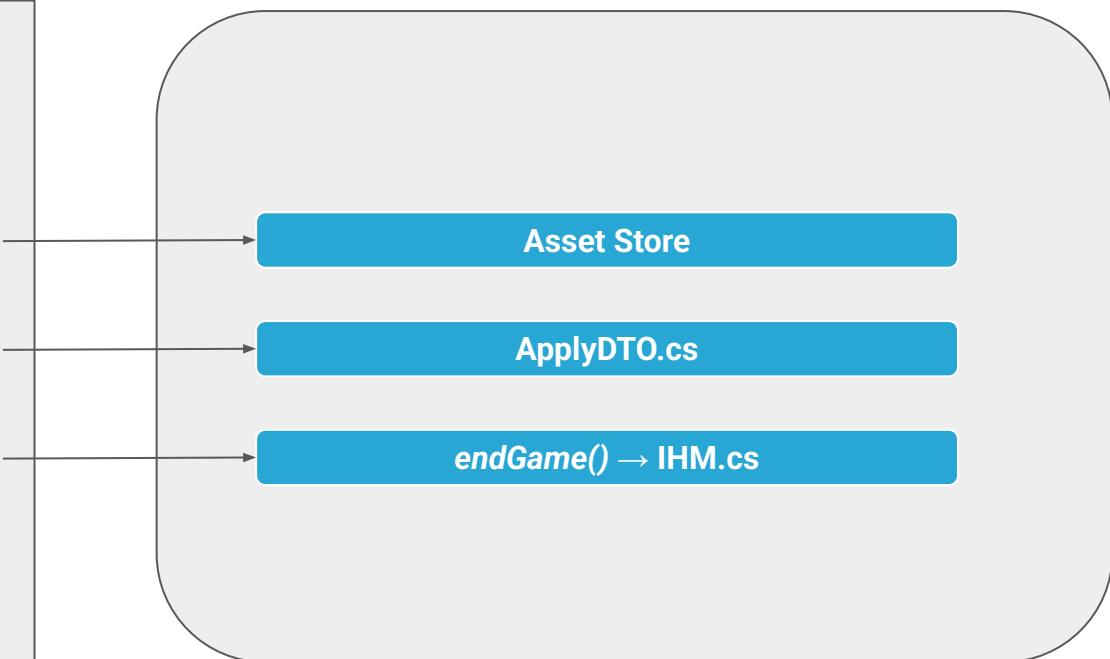
Instantiate

AddComponent



Game**Spécificités**

- Décor médiéval
- Effets de clics + sons
- Feux d'artifices fin de jeu
- Mur matériel

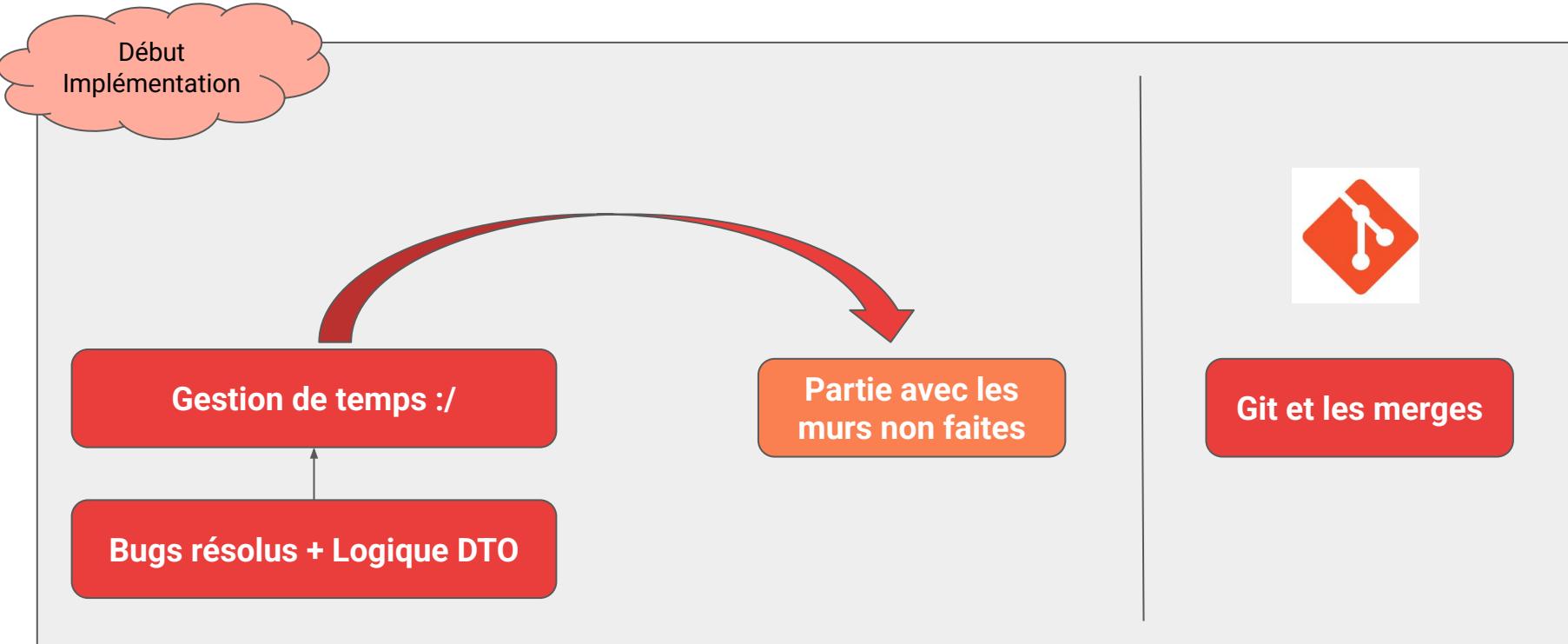


Travail effectué

Présentation des effets



Difficultés



Erreurs d'inattention chemin des assets



Passage de GameObject à Asset pour les Walls

Différence entre Ressources et Prefab

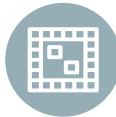


Changement de la matière

Les menus



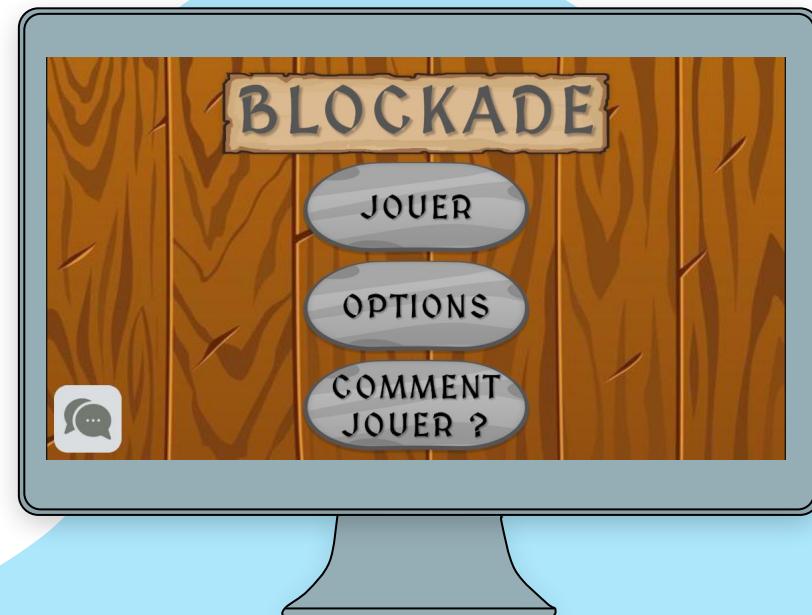
Menus de démarrage



Menu pause/fin de jeu

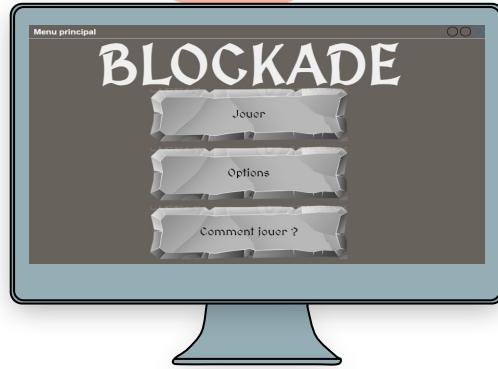


Menu développeur



Menus de démarrage

Changement par rapport au cahier des charges



Jouer

En ligne

Hors ligne

Options

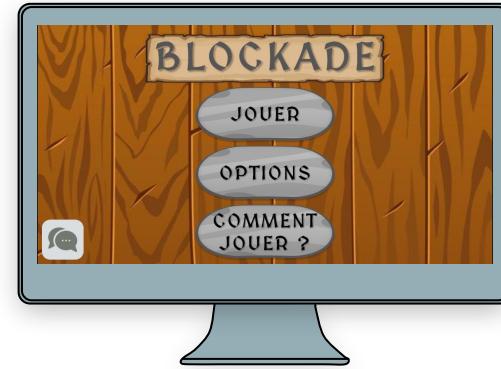
Langue

Volume

Animations

Comment jouer ?

Tuto



Chat

Musique

Connexion

Se connecter

S'inscrire

Jouer

Historique de partie

Options

Volume

Menus de démarrage

Problèmes rencontrés

1

Rect Tool / Move Tool



Move Tool



Rect Tool

2

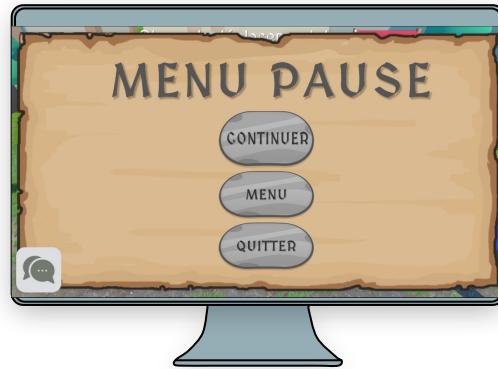
La scroll view

3

Changements par rapport au cahier des charges

Menu Pause

Aperçu du Menu



Pause

Continuer

Menu

Quitter

Menus Pause

Problèmes rencontrés

1

Actionnement du Menu Pause

2

Changements par rapport au cahier des charges

Menu Fin du jeu

Aperçu du Menu



Rejouer

Menu Principal

Médaille dynamique

Menus Fin du jeu

Problèmes rencontrés

1

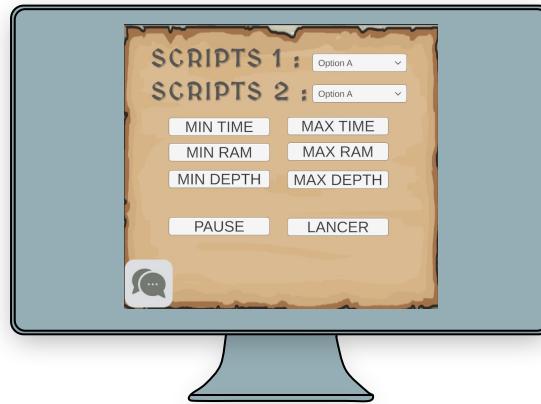
Récupération des informations de la partie

2

Variable selon le résultat

Menu Développeur

Aperçu du Menu



Choix du script

Selection des variables

Mettre en Pause

Menus Développeur

Problèmes rencontrés

1

Affichage du menu

2

Dynamisme des choix

3

Changements par rapport au cahier des charges

Les animations



Gestion des DTO de pions et de murs



Déplacement des pions



Placement des murs



Problèmes rencontrés

1

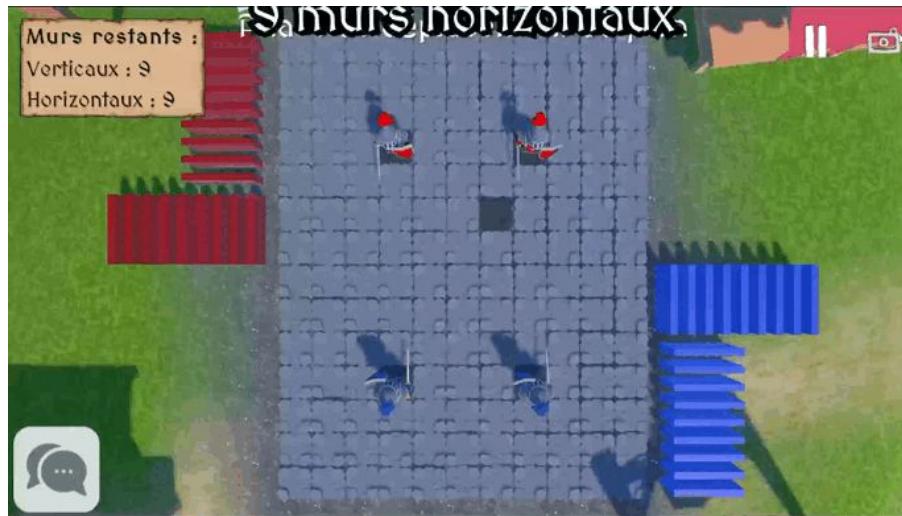
Comprendre le concept des animations

2

Utilisation des Threads

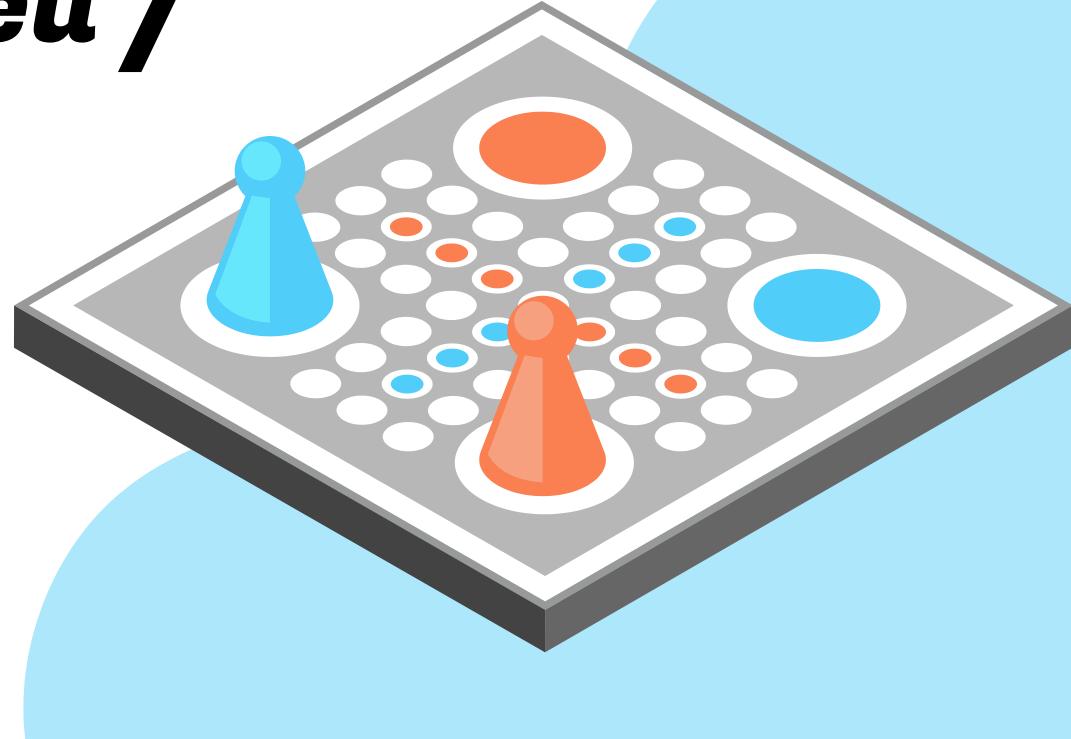
Tâche ajoutée

Réalisation de la vidéo de tutoriel



03

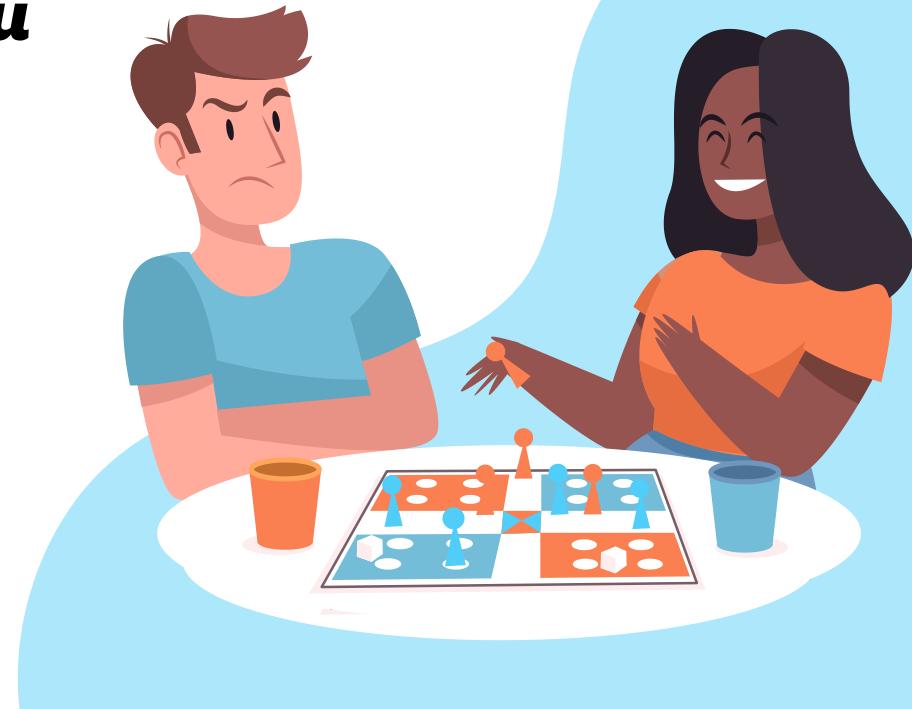
Logique de jeu / Online



Travail effectué

Partie logique de jeu

- Création structure de données
- Création des classes (Square, Player, AlgoBoard)
- Rédaction des fonctions :
 - ◆ D'assignation de cases voisines
 - ◆ D'initialisation d'une partie
 - ◆ De vérification de victoire
- Énumération des types d'erreurs



Patricia

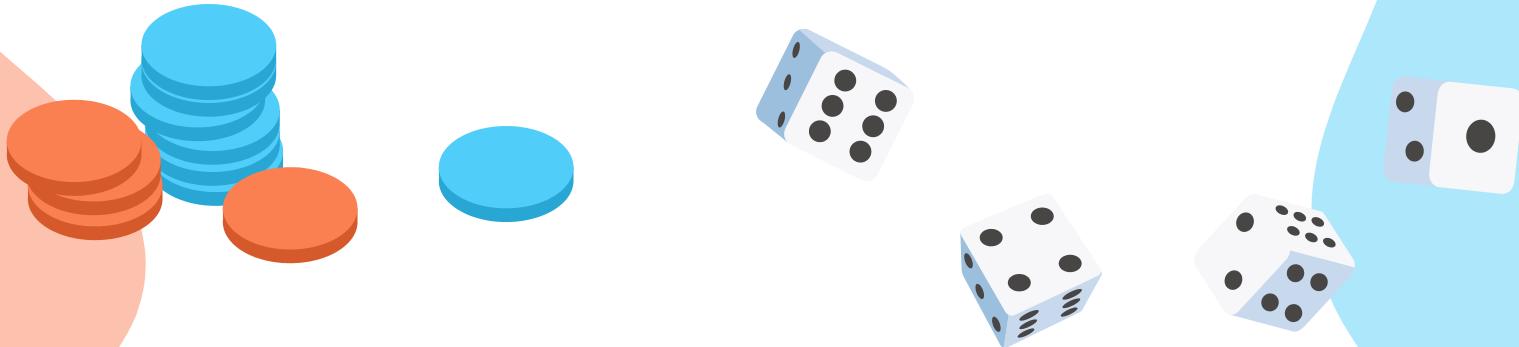


Partie Online

Création des fonctions de conversion de DTO en JSON et de JSON en DTO

Difficultés rencontrées

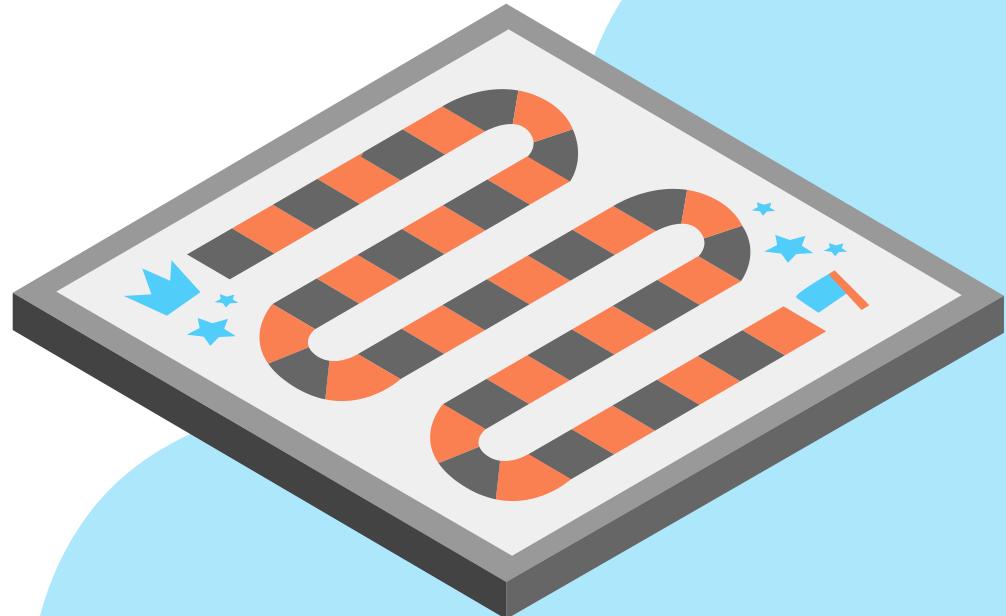
- Difficulté à visualiser la relation entre classes et types de variables
- Utilisation de la fonction de Unity permettant la conversion en JSON



Travail effectué

Logique de jeu

- Création d'une méthode de simulation de la réception d'un DTO de l'IHM
- Création des DTO à retourner à l'IHM :
 - Placement de mur
 - Déplacement de pion
 - Retour des erreurs
 - Retour de l'état de la partie



Travail effectué



Partie Online

- Ajout de la gestion d'invitation de lobby:
 - Envoie d'une invitation à un lobby
 - Connexion à une partie grâce à une invitation
- Ajout de l'envoie d'une demande d'ami

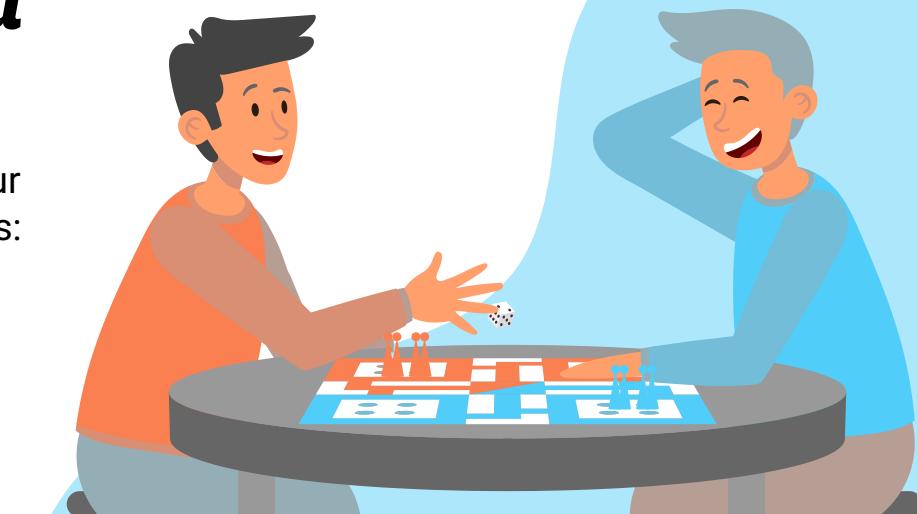
Difficultés rencontrées

- Modifications sur le format de certains DTO
- Premières communications entre Unity et le serveur
- Conversion des json reçus en string sur unity

Travail effectué

Partie logique de jeu

- Création de méthodes essentielles pour les mouvements et actions des joueurs:
 - ◆ isPathPossible
 - ◆ isWallPlaceable
- Création de méthodes pour tester
- Relecture et révision du code



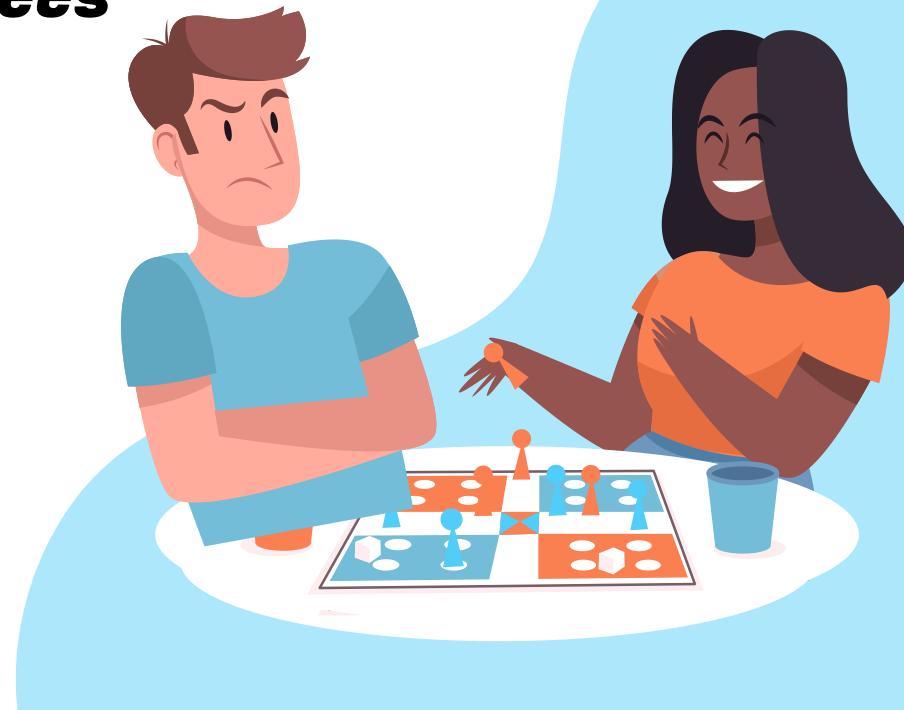


Partie Online

- Développement de méthodes outils sur Unity :
 - JoinLobby
 - HostLobby
 - SendMessageToLobby
 - StartGame
 - endAction

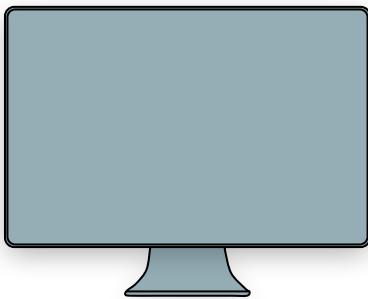
Difficultés rencontrées

- Edges cases pour la méthode isPathPossible
- Homogénéisation du code
- Ajout des librairies WebSockets sur Unity



Travail effectué

Mise en place du serveur WebSocket

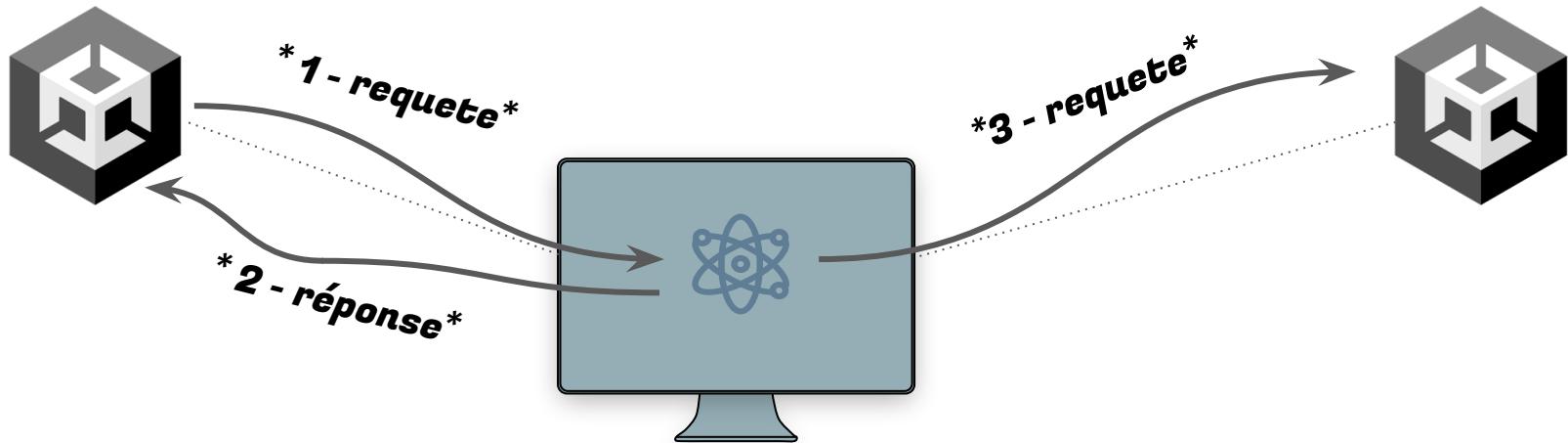


Serveur NodeJS

- 1. Permettre aux clients Unity de s'y connecter (authentification)**
- 2. Associer les clients à des salons, puis des parties**
- 3. Traiter les requêtes des clients selon la situation (créer salon...)**

Travail effectué

Fonctionnement du serveur



Structure JSON: {

token: '0xABCD',
requestType: 'lobby-send-msg',
userId: 2700,
data: 'Hello World'

}

{

result: 200,
responseType: 'lobby-send-msg',
msg: 'Message envoyé!'

}

Difficultés rencontrées

- 1. Identification des connexions***
- 2. Fermeture de salons***
- 3. Gestion des déconnexions***

Trouver chemin

Objectif

Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

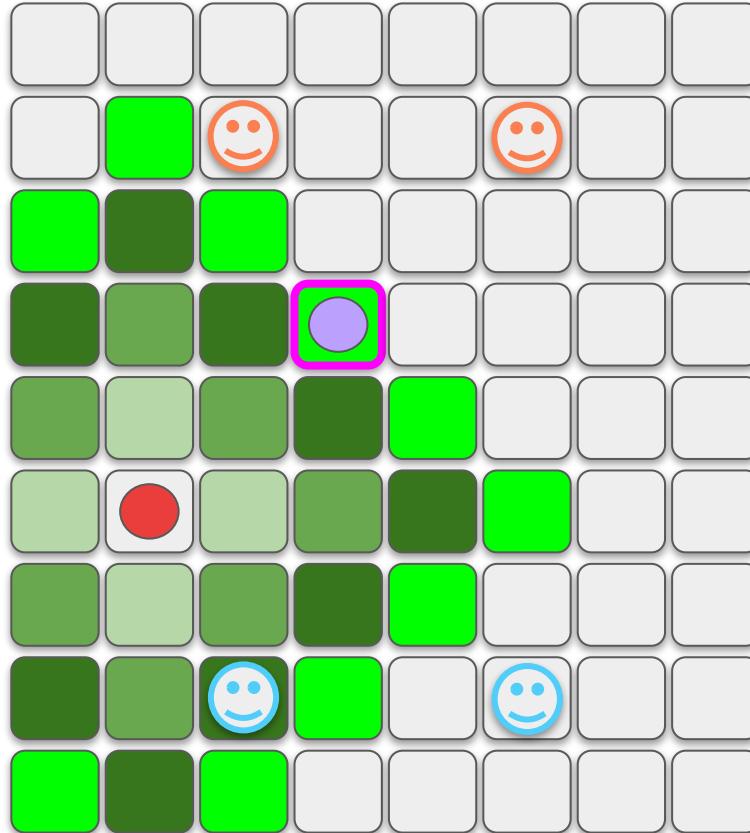
Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

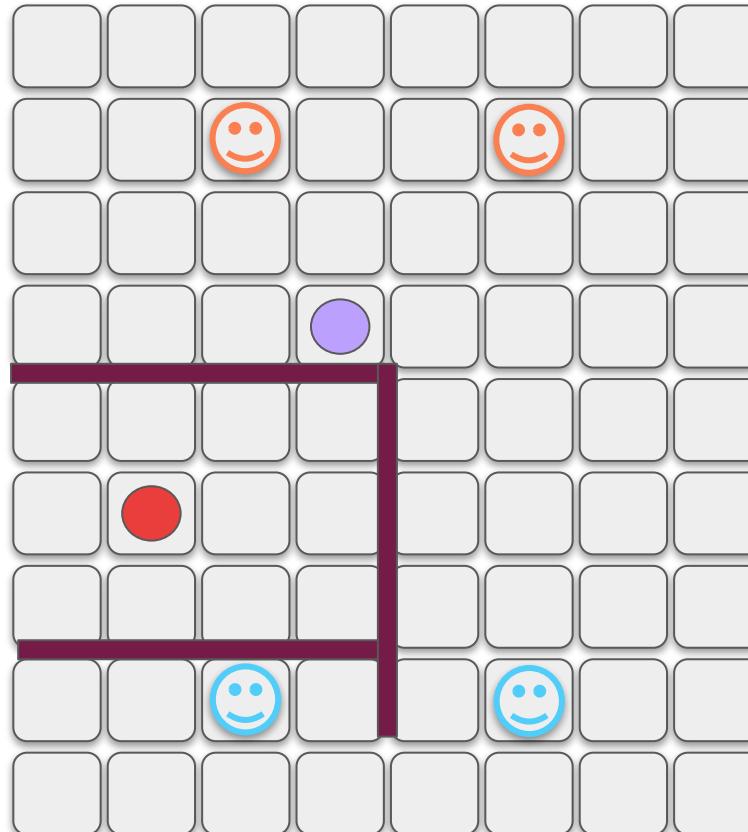
Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

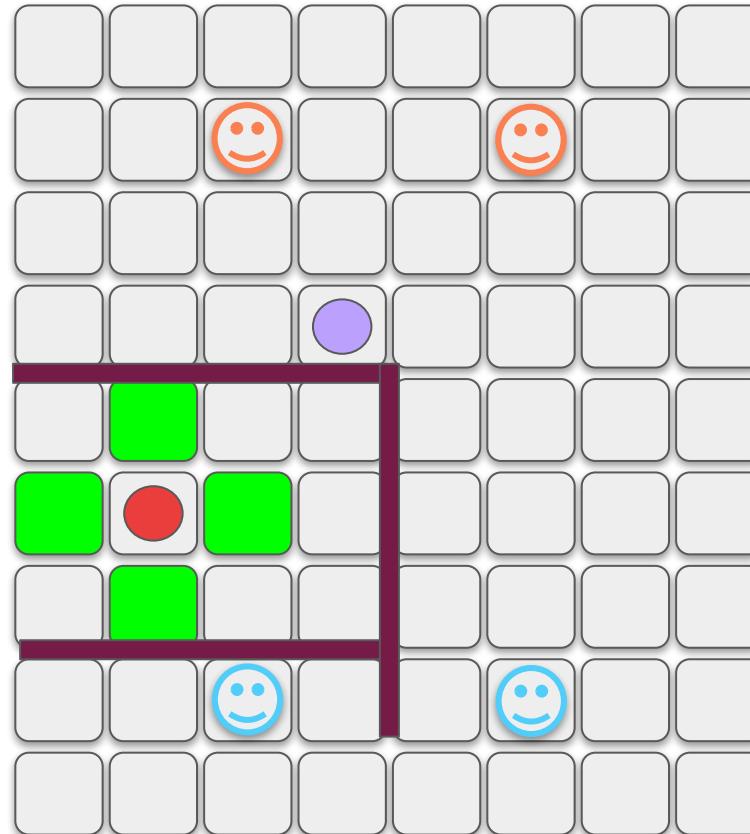
Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

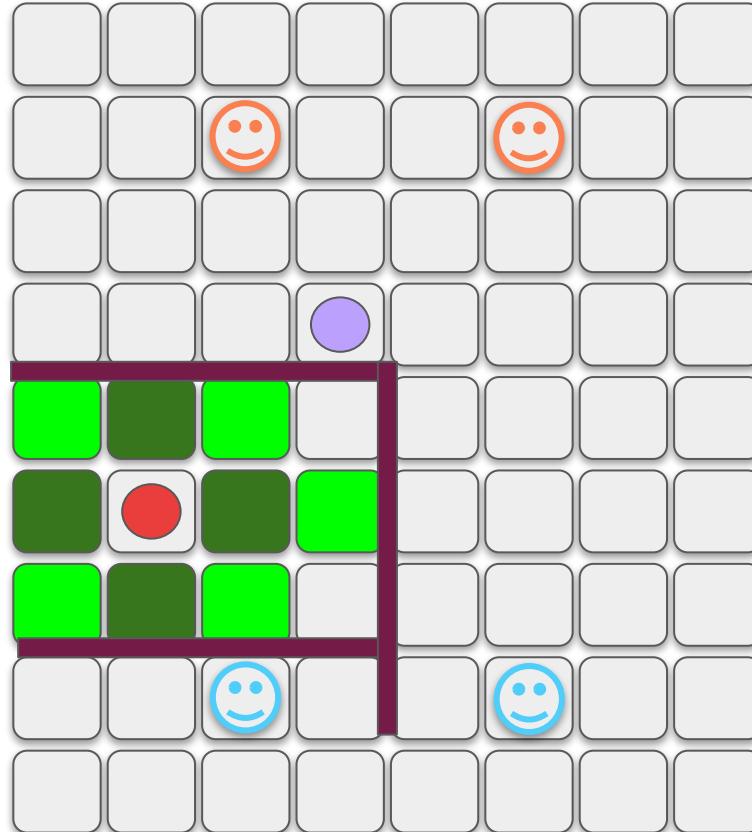
Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin

Objectif

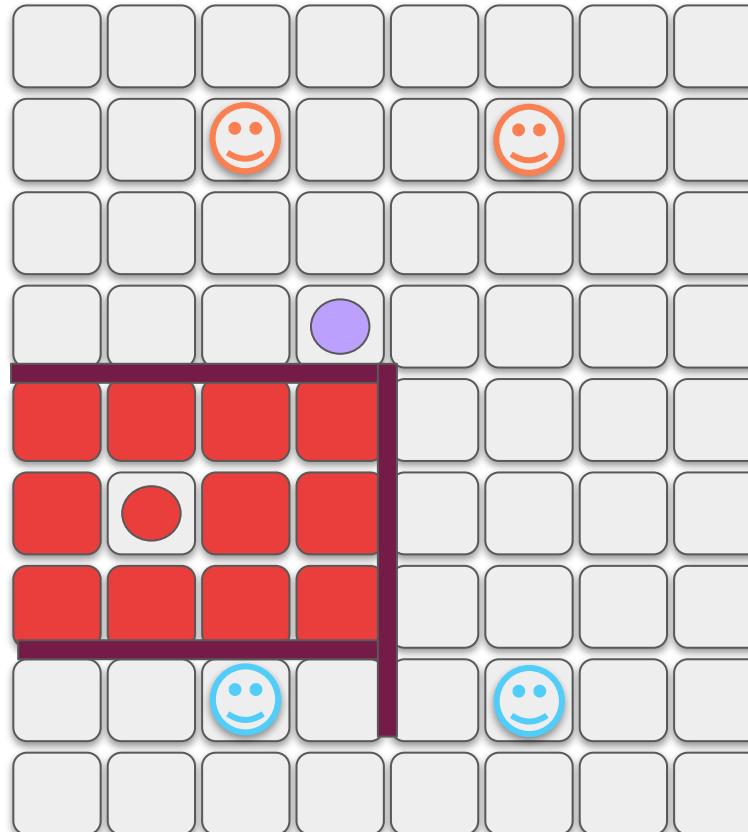
Déterminer si un chemin existe entre un point **A** et un point **B**.



Trouver chemin

Objectif

Déterminer si un chemin existe entre un point A et un point B.



Trouver chemin valide

Objectif

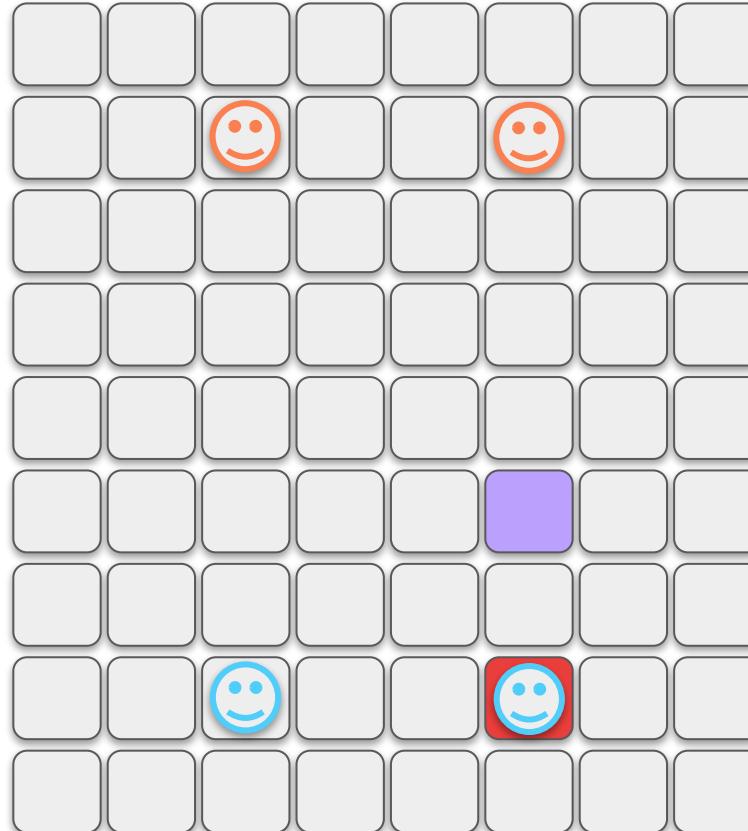
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

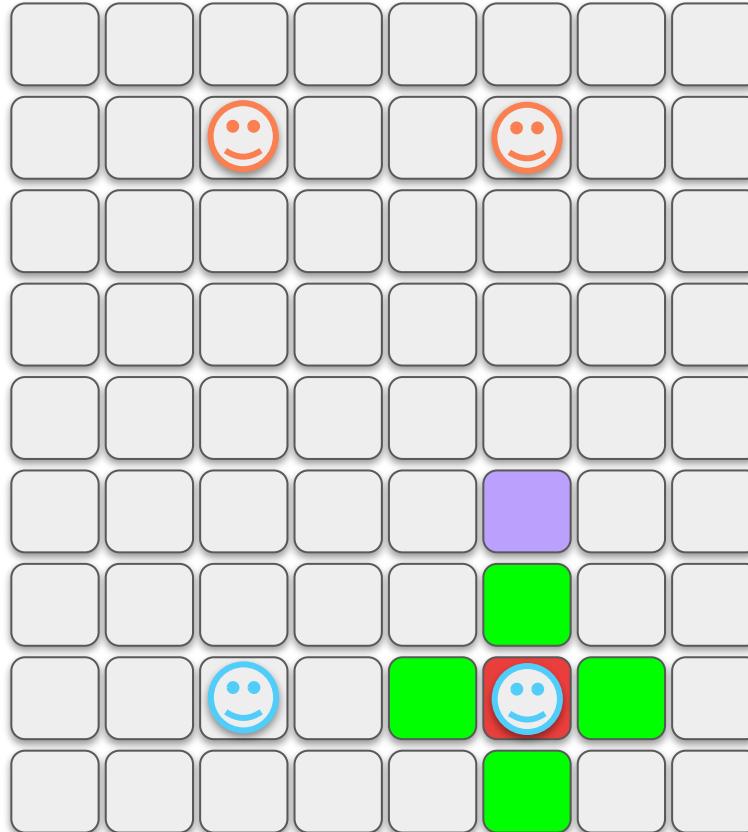
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

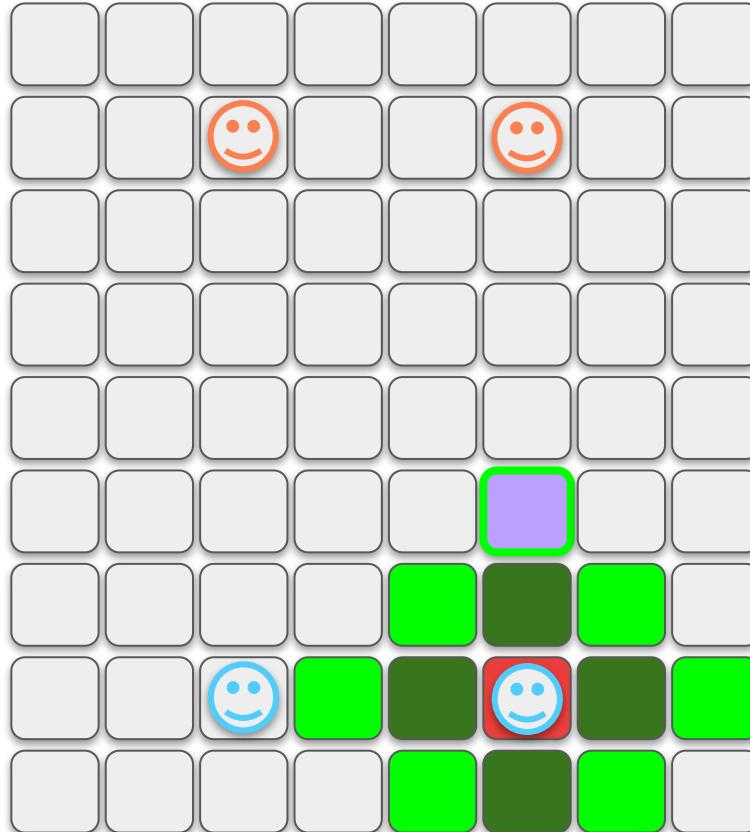
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

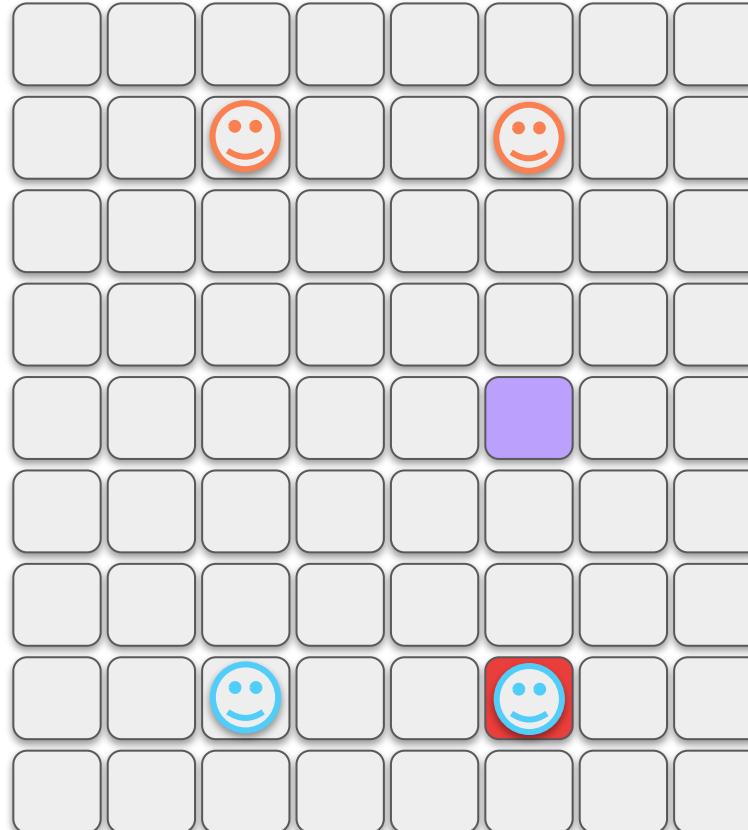
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

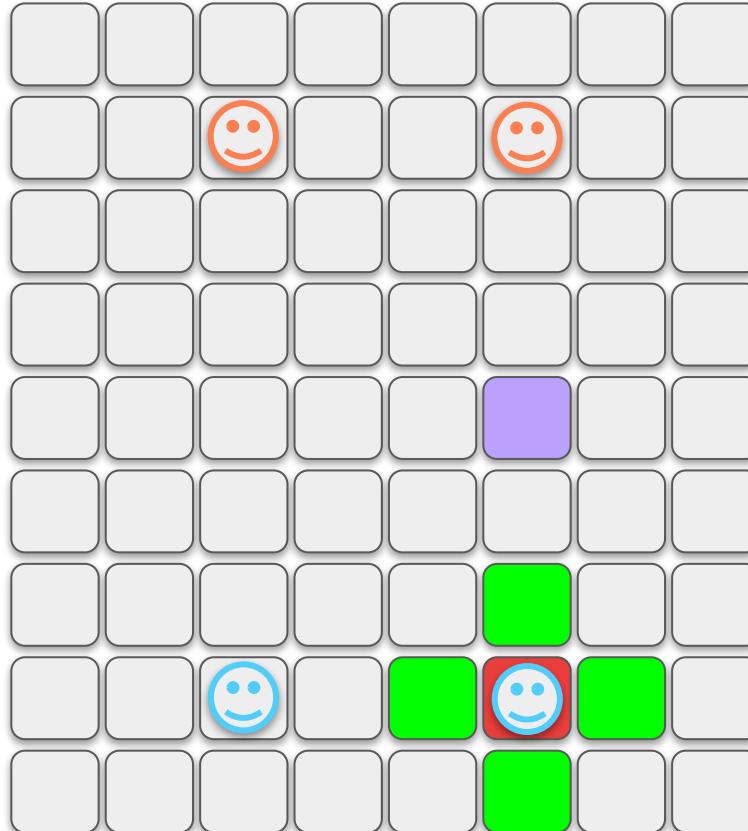
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

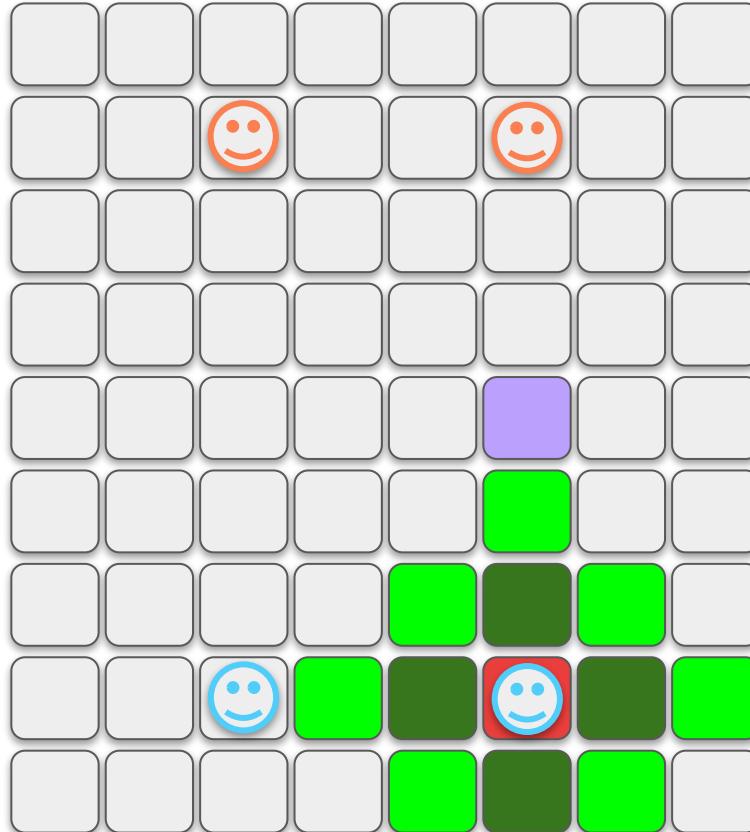
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

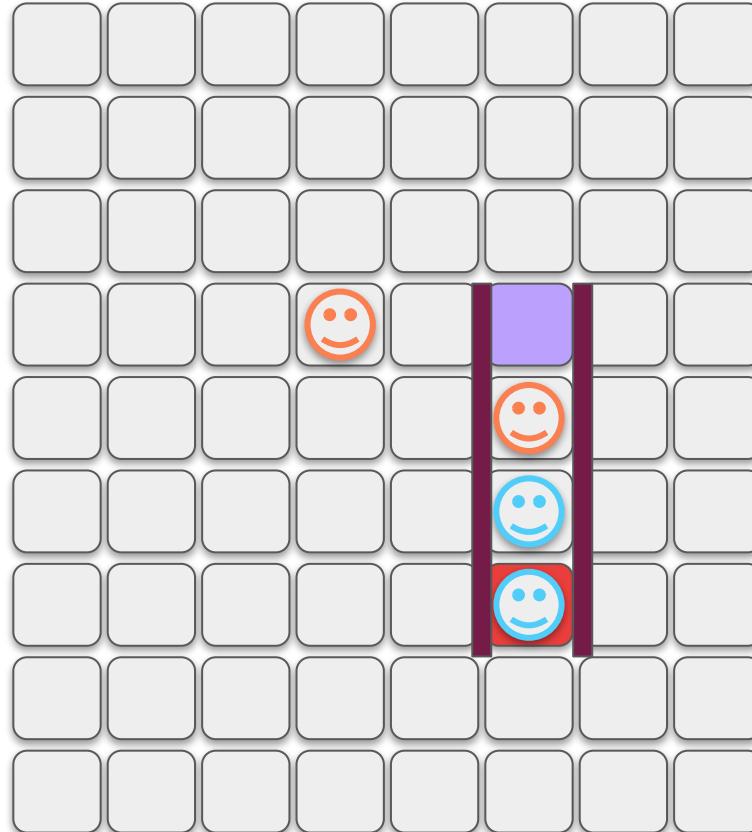
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

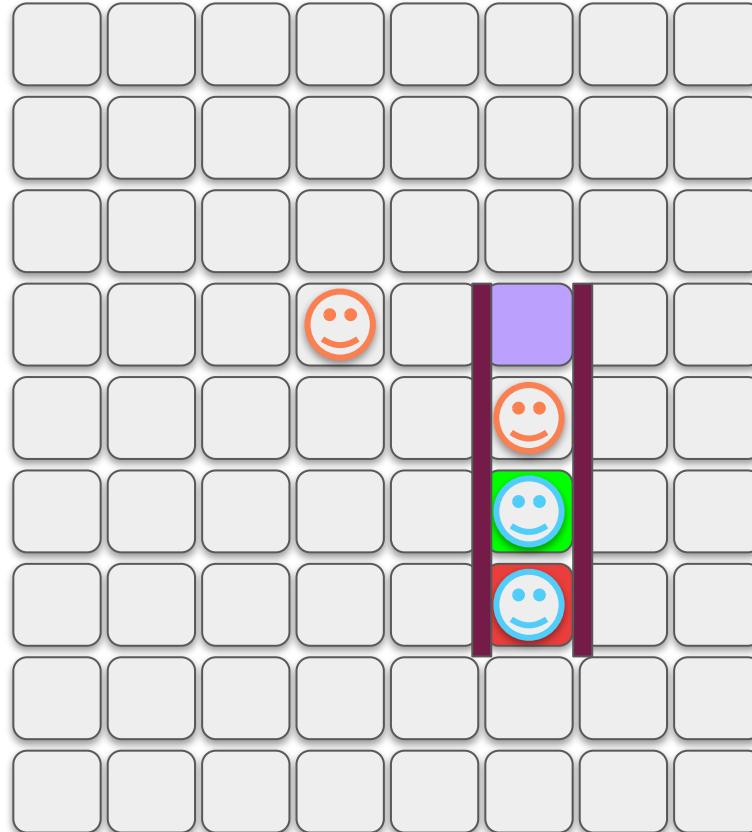
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

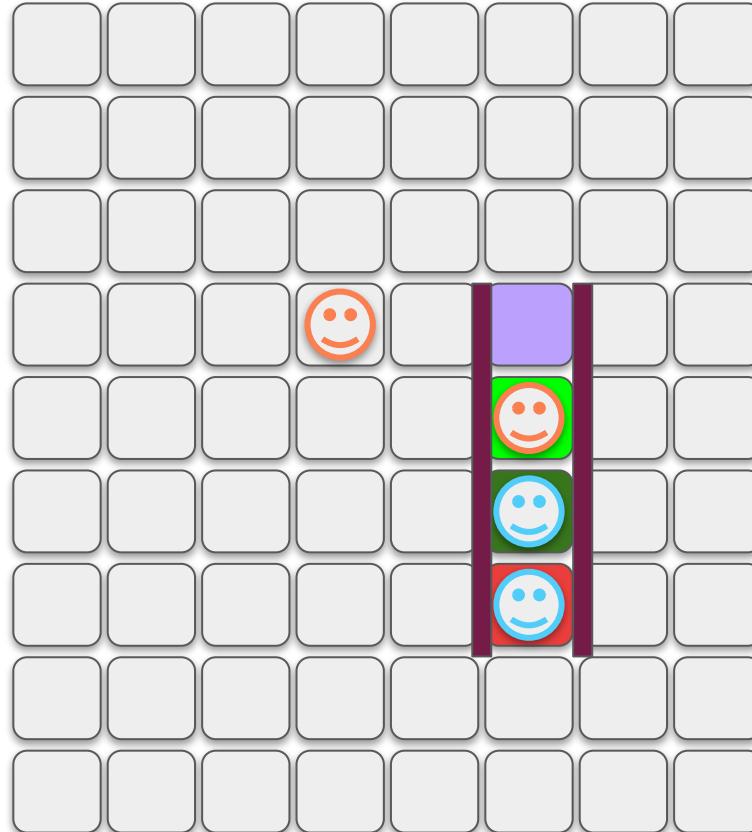
Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



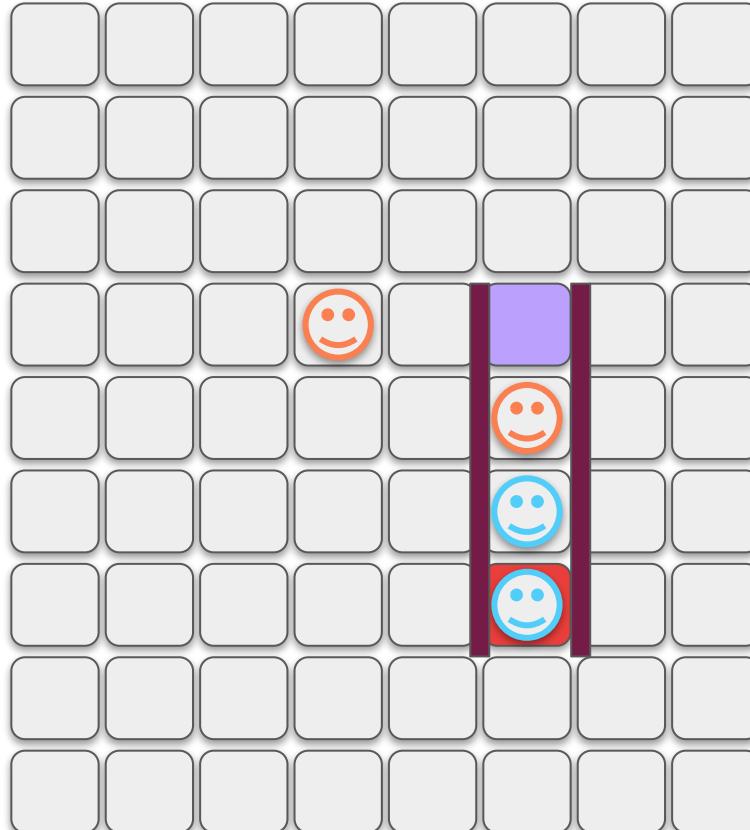
Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.

Etat
pas en
saut

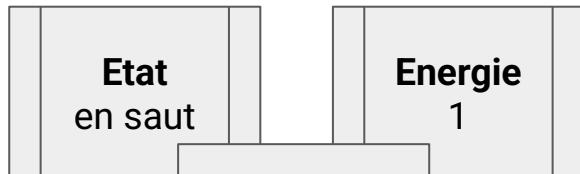
Energie
2



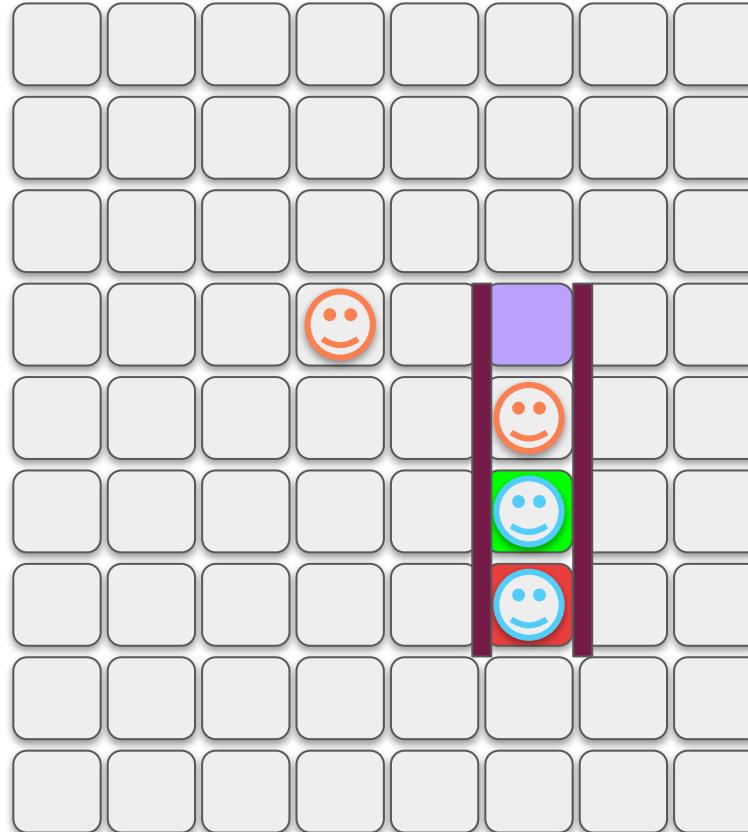
Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



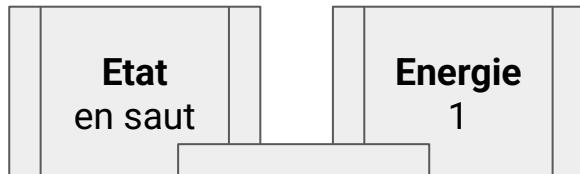
Le passage de l'
état "pas en saut"
à "en saut" coûte
1 énergie.



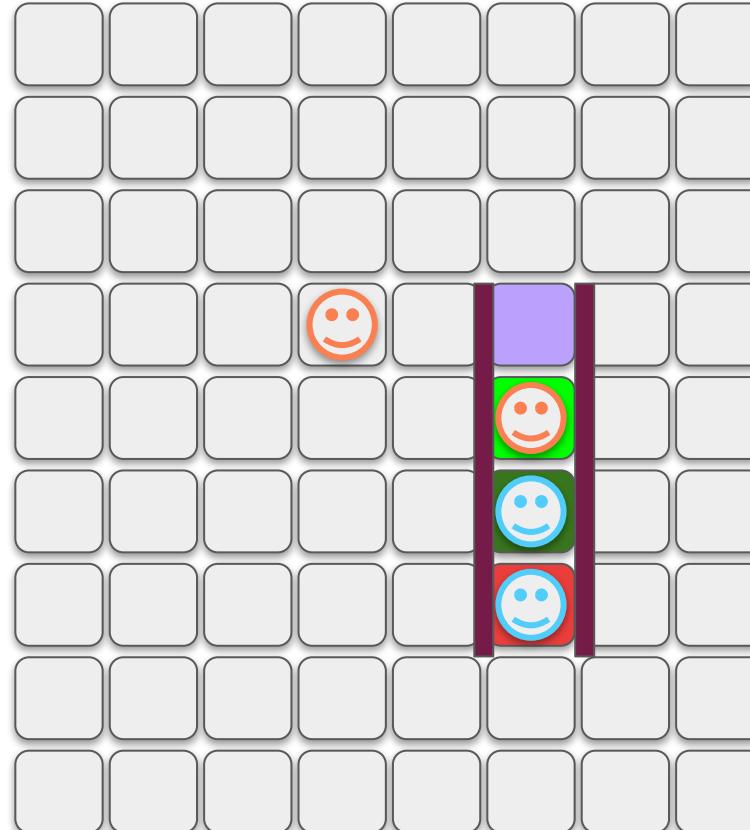
Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



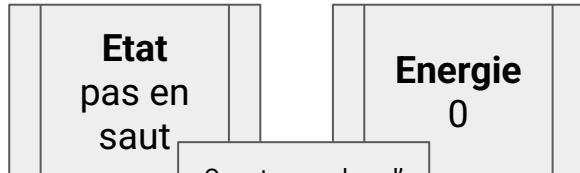
On reste dans l'état "en saut", pas de coût d'énergie.



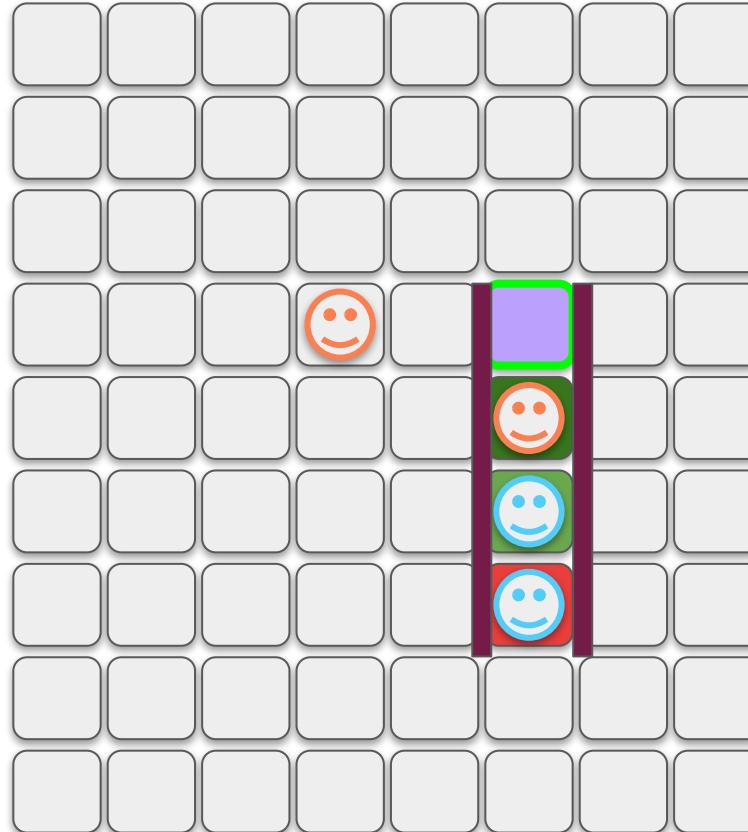
Trouver chemin valide

Objectif

Déterminer si un chemin existe entre un point A et un point B qui respecte les contraintes des règles du jeu.



On retourne dans l'état "pas en saut".
On "atterrit", ça coutume notre dernière énergie.



Logique de gestion de partie

Objectif

Assurer la réception des DTO, leur traitement et vérification, leur validation, et leur transmission aux éléments concernés.

- JcJ Local
- JcJ Online
- JcE

Logique de gestion de partie

1 - Réception

Réception du DTO envoyé par l'IHM ou le Online.

Logique de gestion de partie

2 - Verification du type de DTO

Vérification du type du DTO, soit Wall soit Pawn. Le contenu est celui spécifié dans les échanges entre IHM et Algo dans ce sens là.

Logique de gestion de partie

3 - Vérification de la cohérence et des droits

Vérification du contenu du DTO et de la capacité du joueur à jouer ce type d'action.

Sinon, envoi d'une erreur à l'IHM.

Logique de gestion de partie

4 - Vérification de la légalité de l'action

Vérification de la légalité, à savoir si le déplacement est possible, ou si le placement du mur respect les règles.
Si oui, on transmet le nécessaire à l'AlgoBord et l'IHM pour affichage et prise en compte.
Si non, on envoi une erreur.

Logique de gestion de partie

5 - En fonction du mode de jeu

Si le mode de jeu est JcE, on envoi également le DTO au bot afin de tenir à jour son état du plateau et de la partie.

Si le mode de jeu est Online, on envoi également le DTO via websocket au serveur afin qu'il soit transmis à l'autre joueur.

Logique de gestion de partie

6 - Fin de tour

On tient compte des actions dans le tour en cours, si le DTO reçu termine le tour actuel, on fait le nécessaire pour changer le joueur actif et envoi un DTO de GameState aux partis concernées (Bot, IHM, Online).

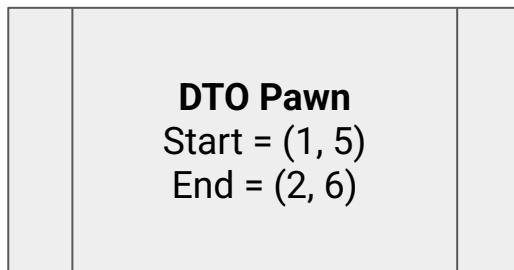
Fonctionnalité d'annulation de move

Objectif

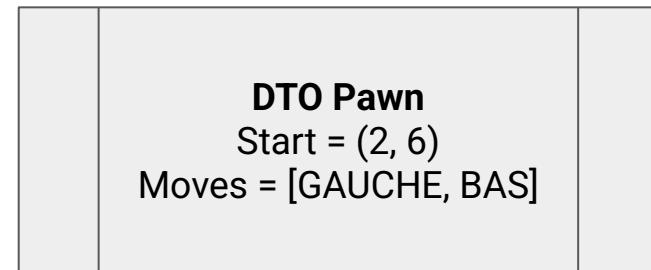
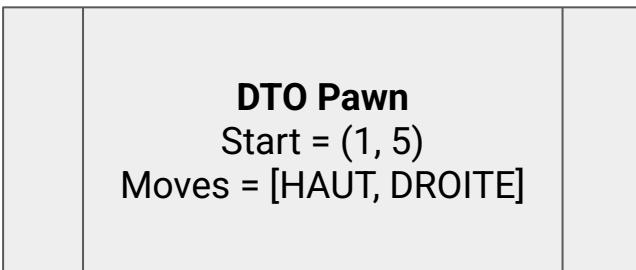
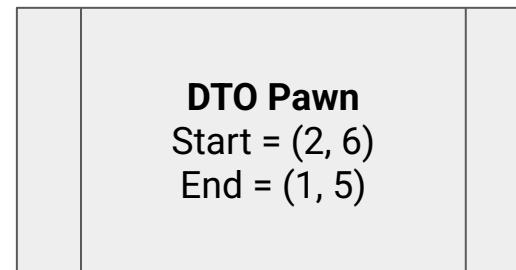
Permettre la possibilité d'annuler un coup.

C'est rendu possible grâce au système de DTO qui sont simple à reforger afin de provoquer le comportement inverse.

Fonctionnalité d'annulation de move

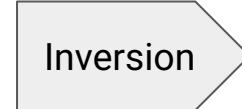


Inversion



Fonctionnalité d'annulation de move

	DTO Wall Start = (1, 5) End = (2, 6) Direction = UP isAdd = true	
--	---	--



	DTO Wall Start = (1, 5) End = (2, 6) Direction = UP isAdd = false	
--	--	--

Fonctionnalité Online

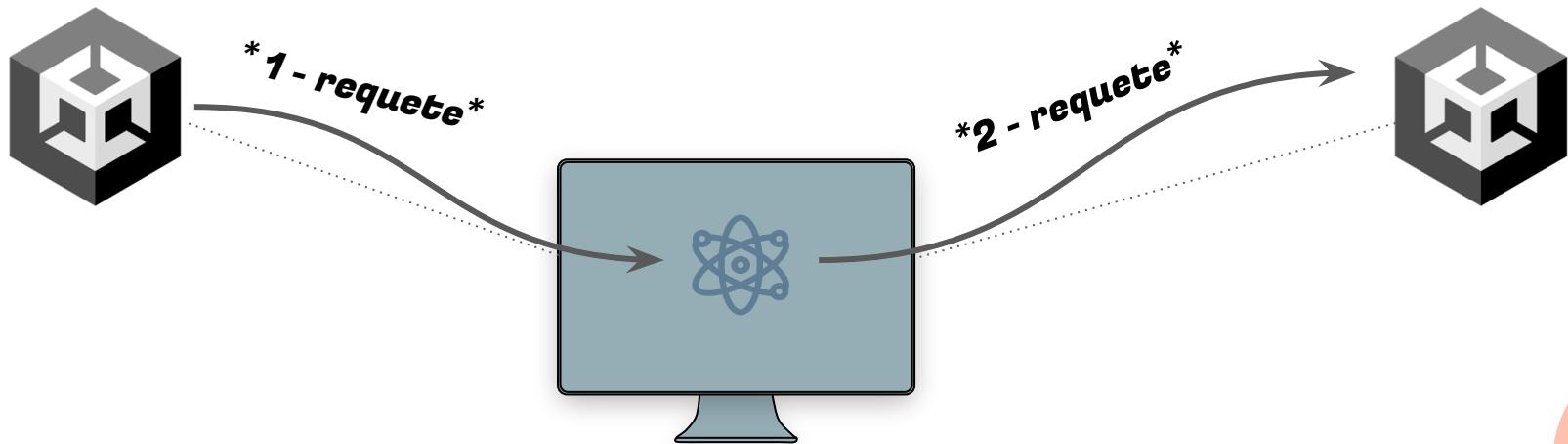
Objectif

Permettre d'établir les communications entre le jeu et l'API ainsi qu'entre les clients via le serveur websocket.

Création d'une structure de donnée et de la BDD ainsi que d'une API robuste en Python.

Fonctionne par l'envoi de requête à l'API pour la création de compte et la connexion notamment.

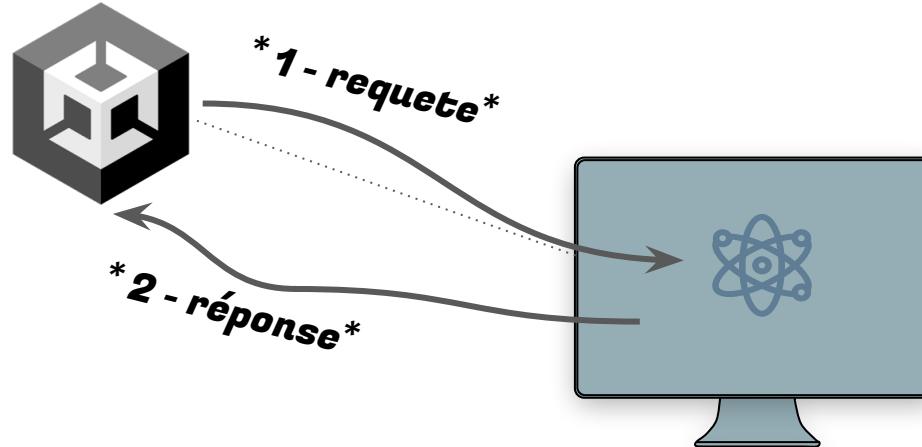
Fonctionnalité Online



Communication WebSocket

Envoi des données, réception puis traitement.

Fonctionnalité Online



Communication API

Développement de l'API en
Python et intégration sur
Unity.

Fonctionnalité Online

Réception

	lobby-host	
--	-------------------	--

	lobby-join	
--	-------------------	--

	lobby-send-msg	
--	-----------------------	--

	lobby-start-game	
--	-------------------------	--

	action-movement	
--	------------------------	--

Action

Interprétation

	Réception du code de la room	
--	-------------------------------------	--

	Un joueur a rejoint la room	
--	------------------------------------	--

	Réception d'un message	
--	-------------------------------	--

	La partie a été lancée	
--	-------------------------------	--

	Réception du DTO de l'action	
--	-------------------------------------	--

Merge

Objectif

Effectuer les Merge fréquemment nécessaire à la fois entre les différentes branches du pôle Logique de jeu/Online, mais aussi les Merge de l'entièreté de la code base.

Merge

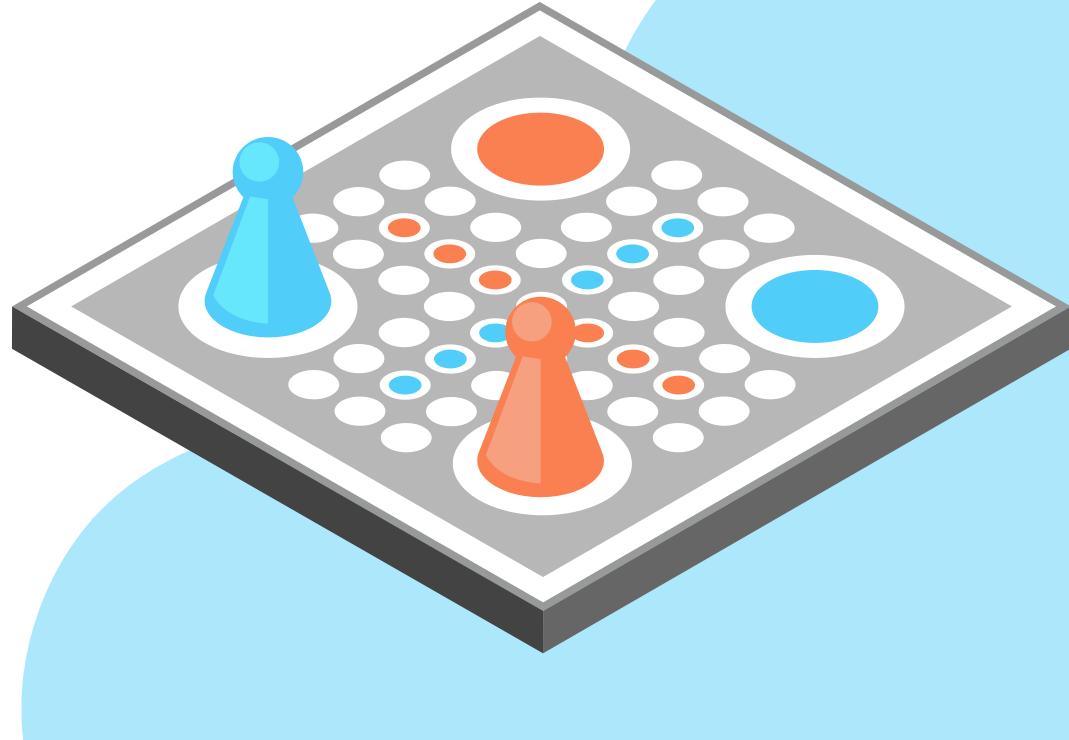
Beaucoup Beaucoup Beaucoup Beaucoup Beaucoup Beaucoup

BEAUCOUP

De bug fix, de modifications, de re factorisation ect.

04

Intelligence artificielle



Guillaume

Framework

Framework basique

Sortie CSV

Mode sans sortie

Guillaume

checkVictory

- *identification condition de victoire*
- *vérification des voisins*

Guillaume

Multithreading

- *intégration du framework*
- *foolproofing*
- *Modularité maximale*

Travail effectué



Data visualization



Suivi de performances



Étalonage des IAs



Technologies



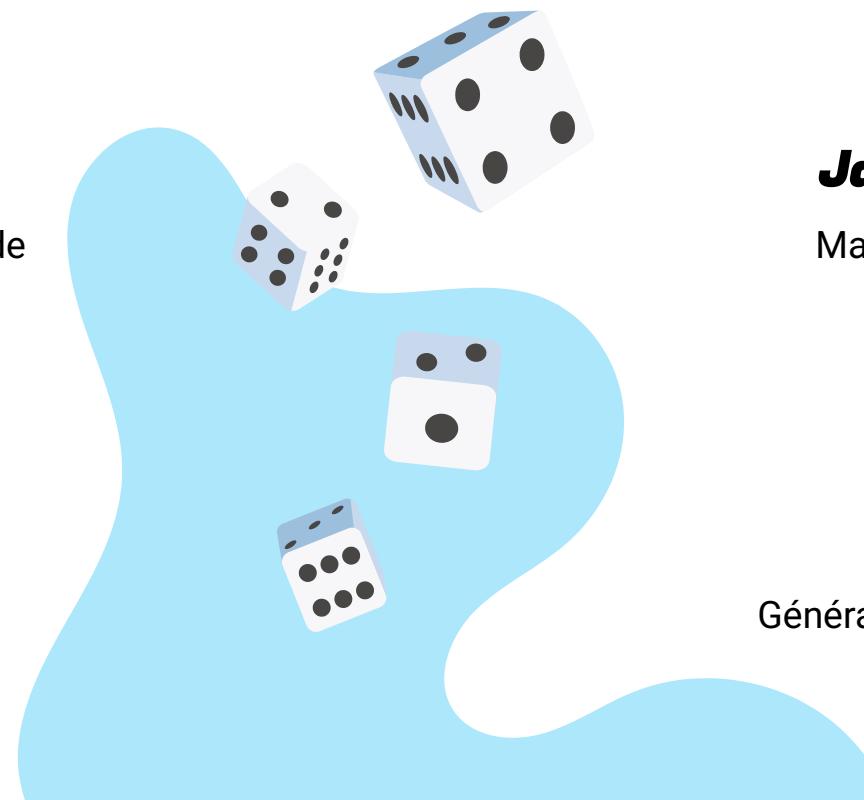
HTML/CSS

Structure et apparence de la page



Apex Charts

Librairie graphiques



Javascript

Manipulation des données



C#

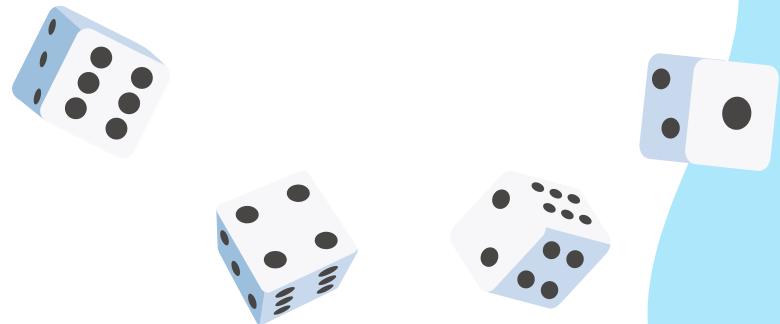
Génération des données

Besoin d'implémentation

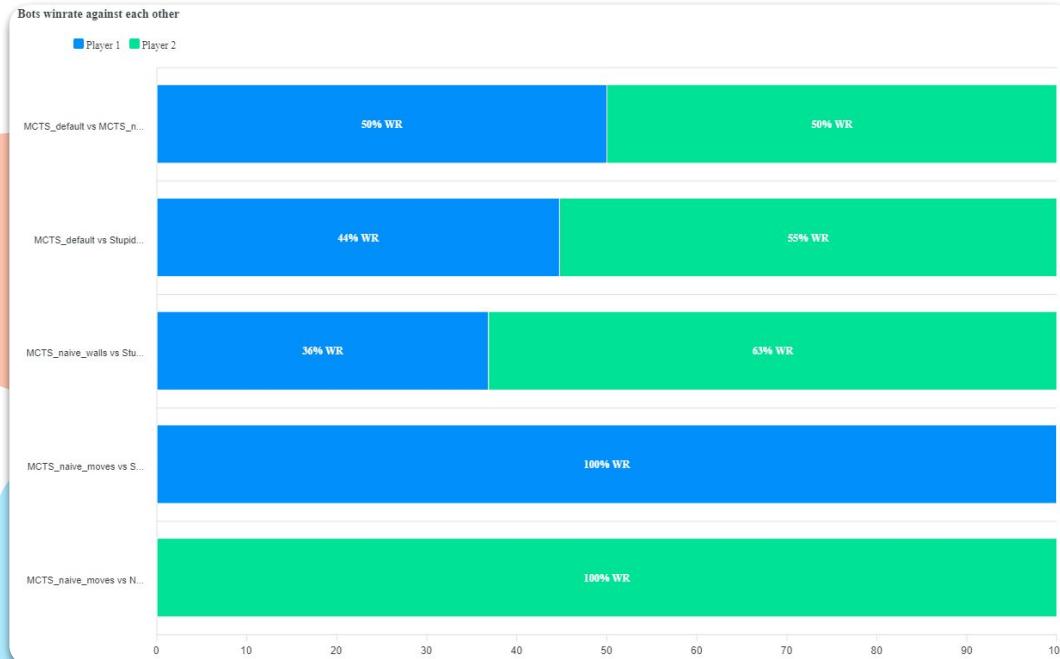
*Suivi de
performances*



Etalonnage



Fonctionnalités



Taux de victoire pour chaque duel

Fonctionnalités

Expected probability of A winning

$$\epsilon_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

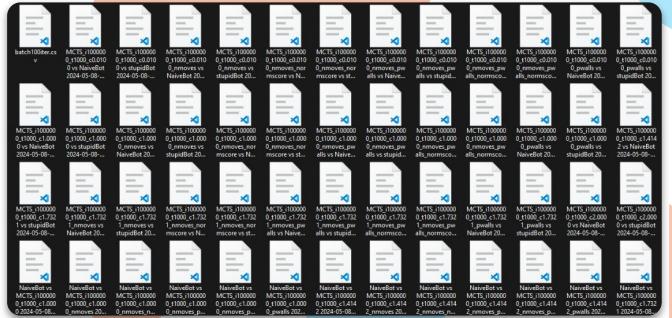
Ratings of both players

Rank	Player	ELO	Wins	Losses	Winrate	Matches	95% CI
1	Naivebot	1337	42	0	100.00%	42	100.00% - 100.00%
2	MCTS_naive_moves	1052	29	42	40.85%	71	29.41% - 52.28%
3	StupidBot	883	45	60	42.86%	105	33.39% - 52.32%
4	MCTS_default	871	36	40	47.37%	76	36.14% - 58.59%
5	MCTS_naive_walls	857	33	43	43.42%	76	32.28% - 54.56%

Établissement d'un classement

Nino

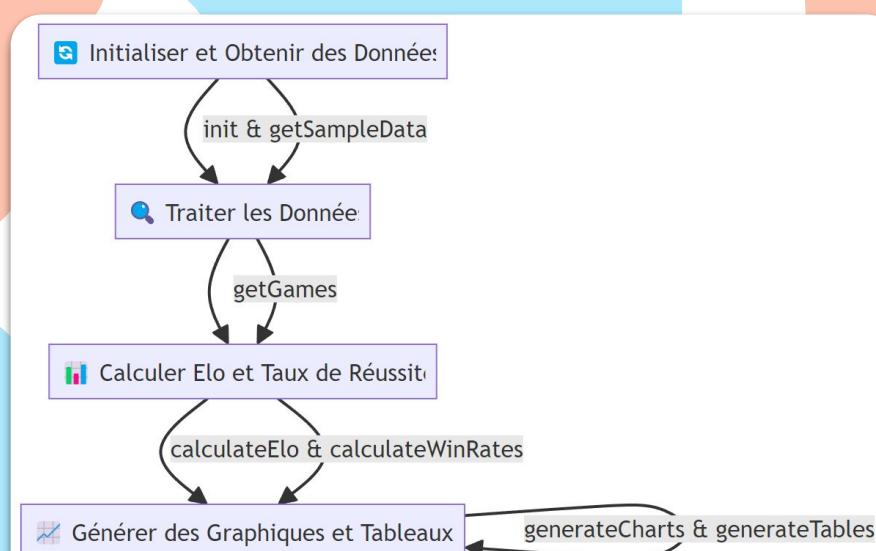
Fonctionnalités

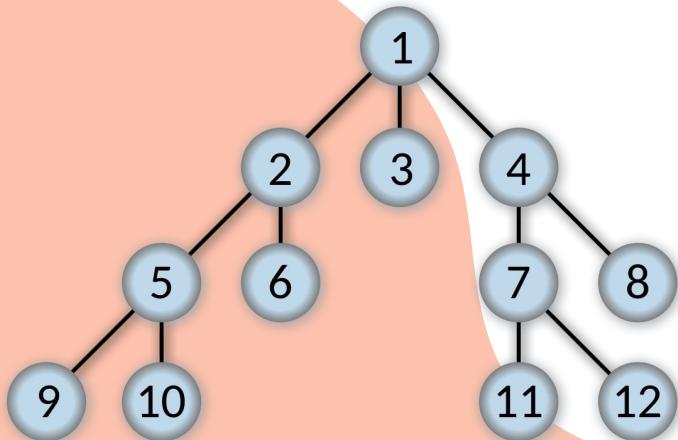


Choose Files | 14 files

Consolidation de données

Fonctionnement





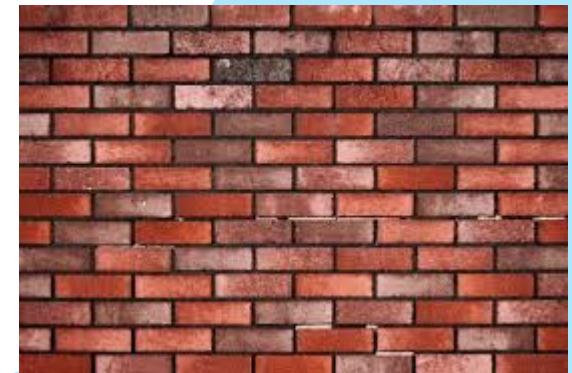
Fonction GetShortestPath()

- **Algorithme BFS**
- **Differents chemins**
- **Prendre celui nécessitant le moins de coups**

Matteo

Fonction GetBestWalls()

- **Tous les placements de mur possibles**
- **Prendre celui qui crée le chemin le plus long**



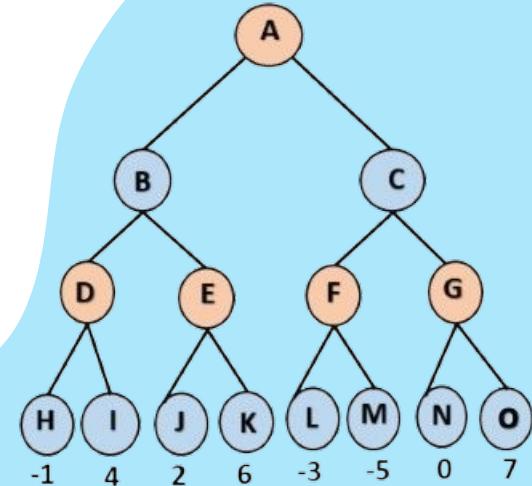
Travail effectué : Bot MinMax

Fonction Minmax() :

Explore les coups pour chaque joueurs

Évalue les états du jeu

Utilisation de l'élagage Alpha-Beta



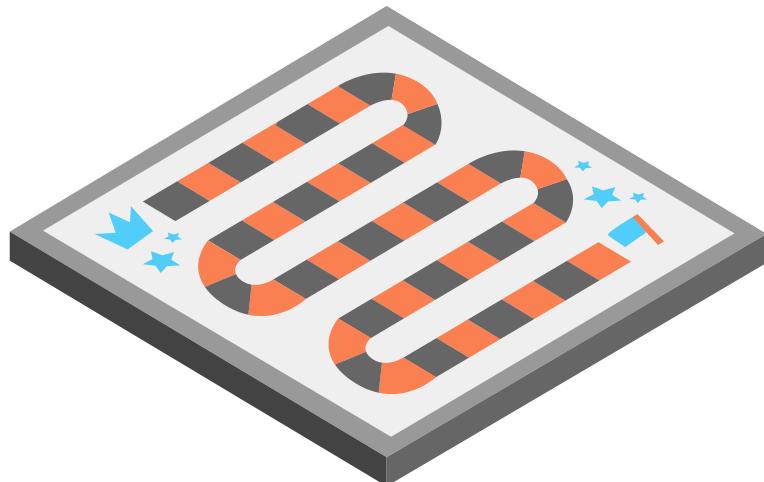
Chloé

Fonction EvaluateBoard()

Calcul un score pour chaque point selon : sa distance a la victoire, la bloquabilité de ses chemins disponibles

Fonction getBlockability()

Évalue la possibilité de bloquer une position avec un mur



Chloé

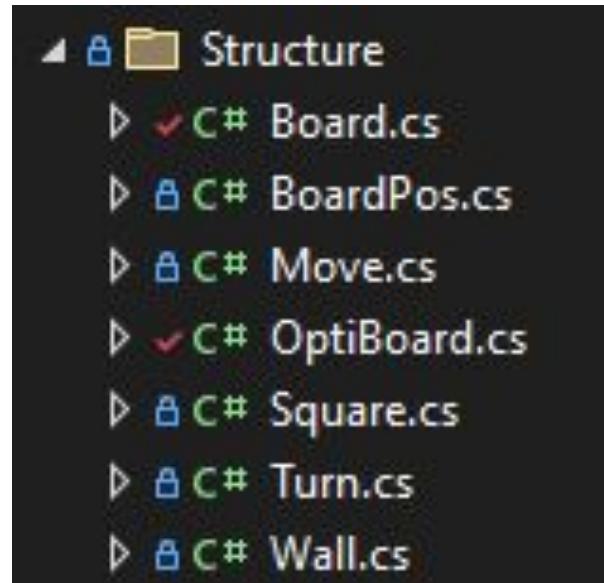
Difficultés

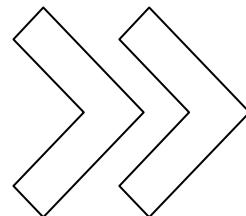
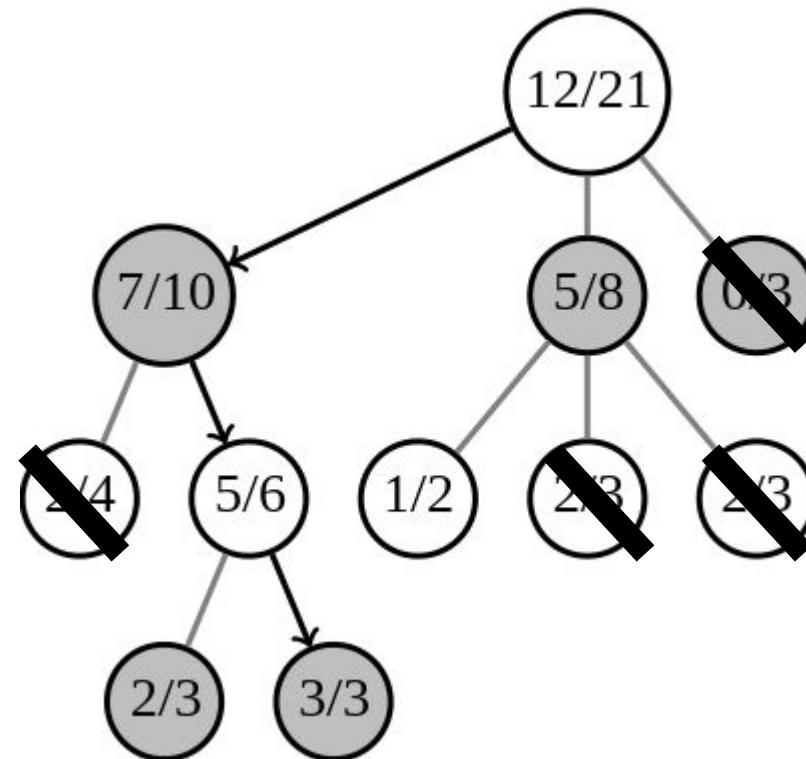


Heuristique non définie

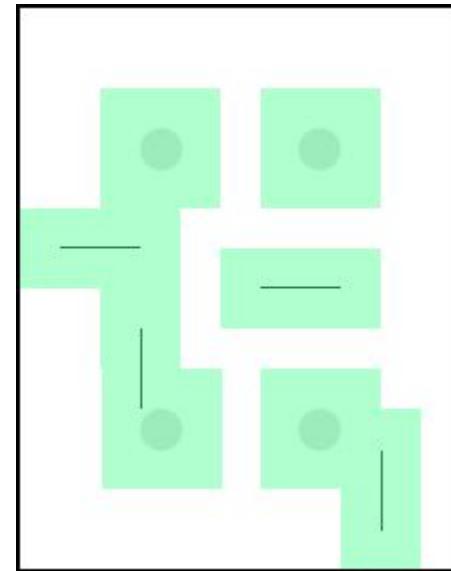
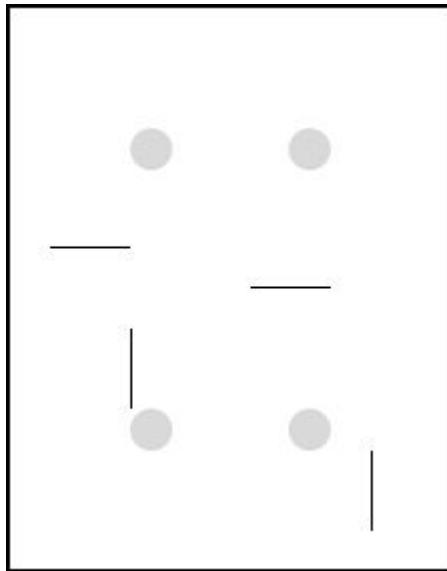


Optimisation des performances

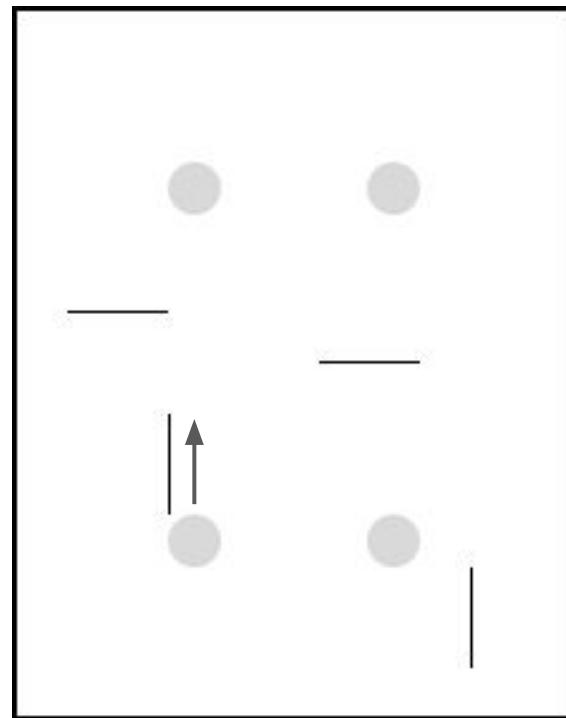


Mode difficile: Recherche arborescente de Monte-Carlo**Optimisations****Heuristiques**

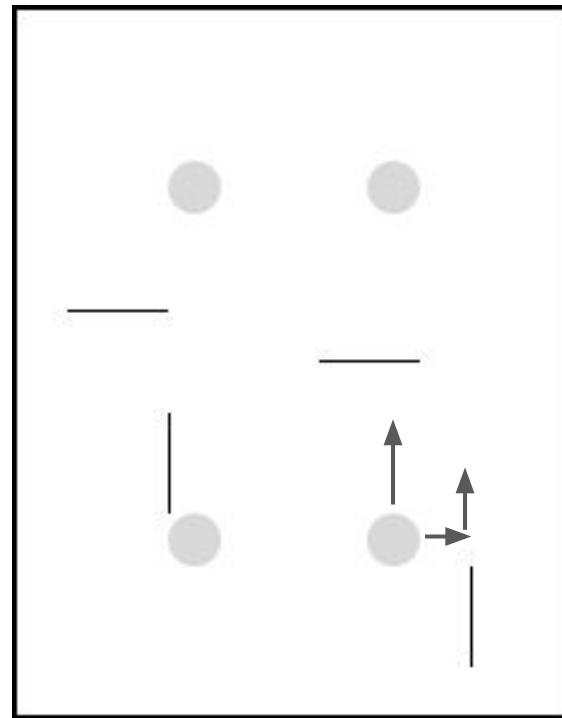
1. *Les murs prometteurs*



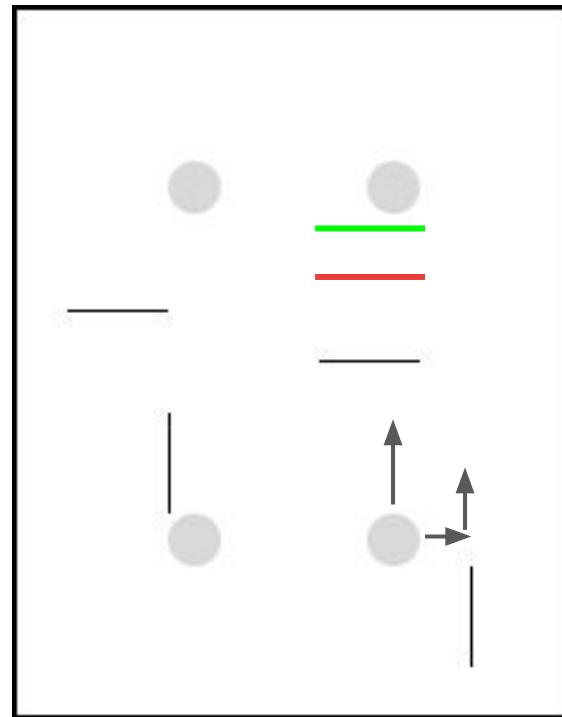
2. Les coups naïfs



3. “Naïve Boosted”



3. “Naïve Boosted”

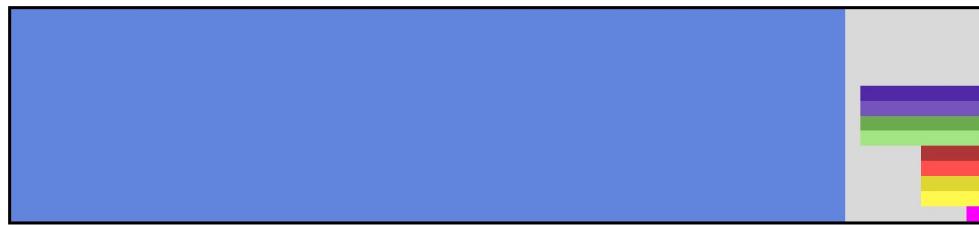


1. Les optimisations de la structure: bitflags et bitboards

```
public int surroundedBy;  
public const int WALL_ABOVE = 0b1000;  
public const int WALL_RIGHT = 0b0100;  
public const int WALL_BELOW = 0b0010;  
public const int WALL_LEFT = 0b0001;
```

```
4 références  
public bool hasWall(int wall_bitflag)  
{  
    return (surroundedBy & wall_bitflag) == wall_bitflag;  
}  
  
4 références  
public void addWall(int wall_bitflag)  
{  
    surroundedBy |= wall_bitflag;  
}
```

1. Les optimisations de la structure: bitflags et bitboards



Squares data



isYellowPlaying



yellow vertical
walls left



yellow horizontal
walls left



red vertical
walls left



red horizontal
walls left



yellow pawn1
position



yellow pawn2
position



red pawn1
position



red pawn2
position

2. *Les optimisations du pathfinding*

Greedy first search

- Génération des coups legaux

A*

- CheckVictory
- Heuristiques MCTS

3. Optimisations diverses

- Copie de la memoire
- Ne pas générer les plateaux jusqu'à en avoir besoin
- MCDagS + Zobrist hashing (RIP)
- Verification des murs adjacents aux murs existants
- Ordre des vérifications dans getLegalWalls pour faire le moins de pathfinding possible
- Pré-calcul de l'heuristique de distance pour A*/greedy

***Merci pour votre
attention !***