

---

# PROJET : JEU DU BOBAIL

---

Hippolyte PAILLER et Aliona LEJUSTE

# SOMMAIRE

Introduction	1
Affichage	2
Déplacements	3
Victoire	4
Main	5
Implémentation des IAs	6
Conclusion	7

# INTRODUCTION

Le Bobail est un ancien **jeu traditionnel africain** dans lequel on essaye de ramener le pion central, appelé **Bobail**, dans son camp. Les pions ne peuvent pas se sauter mais servent d'obstacles.

# 01

# AFFICHAGE

Plateau et pions

Le plateau fait **5 cases** de côté et chaque joueur place **5 pions de son côté** (sur les cases au bord du plateau). Un pion appelé Bobail, est placé sur la case située au **centre du plateau**.

# O2

# AFFICHAGE

```
#ifndef COULEUR_PION_H
#define COULEUR_PION_H

typedef enum {
    BLANC,
    BLEU,
    ROUGE,
    VERT
} CouleurPion;
```

	0	1	2	3	4
0	●	●	●	●	●
1	.	.	.	.	.
2	.	.	●	.	.
3	.	.	.	.	.
4	■	■	■	■	■

## AFFICHAGE EN UNICODE AVEC UN CODE COULEURS

Nous avons décidé de prendre un modèle **bataille navale**.

```
CouleurPion** creer_grille(int lignes, int colonnes) {
    // Allouer de la mémoire pour le tableau de pointeurs
    CouleurPion** grille = (CouleurPion**)malloc(lignes * sizeof(CouleurPion*));
    // Allouer de la mémoire pour chaque tableau d'entiers
    for (int i = 0; i < lignes; i++) {
        grille[i] = (CouleurPion*)malloc(colonnes * sizeof(CouleurPion));
    }
}
```

## Création de la matrice grille

```
void afficher(CouleurPion** grille){
    printf("\n");
    // la grille est représentée par un tableau d'entiers, il faut convertir ces données
    for (int i=0;i<5;++i){
        for (int j=0;j<5;++j){
            if (grille[i][j]==0){
                printf(". ");
            }
            if (grille[i][j]==1){
                printf("O ");//le joueur sera représenté par des O
            }
            if (grille[i][j]==2){
                printf("X ");//l'ordinateur sera représenté par des X
            }
            if (grille[i][j]==3){
                printf("B ");//l'ordinateur sera représenté par des X
            }
        }
    }
}
```

## Ancien affichage simple avec des X et des O pour les joueurs et un B pour le Bobail

# DÉPLACEMENTS

Les déplacements

Les joueurs jouent à tour de rôle.

A chaque tour, le joueur déplace d'abord le **bobail** puis un de ses **pions**.

Le bobail se déplace **dans tous les sens d'une seule case à la fois**.

Les pions se déplacent également **dans toutes les directions mais ils doivent aller au bout de leur déplacement** (jusqu'au bord du plateau ou jusqu'à rencontrer un autre pion ou le Bobail).

# 03

# POSITION\_BOBAI

Décélérer toutes les erreurs possibles :

- la case donnée ne contient pas de pion
- la nouvelle position est hors plateau
- la nouvelle position est déjà prise
- le déplacement est autorisé

```
if (grille[i][j]==VERT){  
    // verifier la possibilité du mouvement
```

```
if (grille[ligne][colonne]==BLANC){//verif si presence d'un pion
```

# GESTION D'ERREUR DU SCANF

```
int ret = scanf("%d %d",&ligne, &colonne);  
//gestion des erreurs de type en entrée  
  
if (ret != 2) {  
    printf("Format invalide. Veuillez saisir deux entiers.\n");  
    // Pour vider le tampon d'entrée en cas de saisie incorrecte  
    while (getchar() != '\n'); // Vide le tampon jusqu'à la fin de ligne  
    position_bobai(grille,joueur);  
    return grille;  
}
```

# POSITION\_PION

01. Déplacement jusqu'à un obstacle ou la limite du plateau. Il faut donc d'abord vérifier que le mouvement est soit **diagonal**, soit **horizontal** soit **vertical**.

```
int decalage_ligne = ligne_position-ligne_pion;  
int decalage_colonne = colonne_position-colonne_pion;
```

```
if (decalage_ligne==0){ //mouvement que selon l'horizontal
```

```
else if (decalage_colonne==0){ //mouvement que selon la verticale
```

```
else if (decalage_colonne == decalage_ligne){//dep en diagonale
```

02. Vérifier que le pion n'a rencontré **aucun pion** avant d'atteindre sa nouvelle position

03. Vérifier que la case d'après contient soit un **pion** soit la **bordure**.

# VICTOIRE

Le but du jeu

- Ramener le Bobail sur une case où se trouvaient ses propres pions au début.
- Encercler le Bobail afin que l'adversaire ne puisse pas le déplacer au début de son tour.

# 04



# MAIN

Le début de partie

Le joueur qui commence la partie,  
ne déplace pas le Bobail. Il déplace  
uniquement un de ses pions.

# 05

# MAKEFILE ET HEADERS

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <unistd.h> // Pour la fonction sleep()
#include "affichage.h"
#include "position_bobai.h"
#include "case.h"
#include "position_pion.h"
#include "victoire.h"
#include "ordi_alea.h"
#include <time.h>
#include "MCTS.h"
```

Début du main

## HUMAIN VS HUMAIN

- switch car case 2 : **humain vs machine**
- on fait appel aux différentes fonctions (**WHILE**)
- 1er tour le bobail ne bouge pas

```
switch(choix){
    case 1:
        // on pourrait demander le nom des joueurs ?

        // Créer la grille de jeu

        joueur = BLEU;
        CouleurPion victory = BLANC;
        afficher2(grille);
        position_pion(grille,joueur); //premier tour,
        afficher2(grille);
        joueur=ROUGE;
```

## VICTOIRE

```
victory=victoire(grille,joueur);
if (victory == ROUGE){
    printf("Victoire du joueur Rouge\n");
    break; }
else if (victory == BLEU){
    printf("Victoire du joueur Bleu\n");
    break;
}
```

# IMPLÉMENTATION DES IA

L'implémentation des IA s'est faite en **3 étapes**. Nous avons construit 3 niveaux d'intelligence artificielle.

La première : **IA en aléatoire**

- Les coups sont choisis par rand();
- IA de base dont les codes ont été utiles pour construire les deux autres.

La deuxième : **IA\_up**

- On simule des parties à partir de certaines configurations de départ;
- Calcul du taux de victoire pour le choix de la configuration à renvoyer.

La troisième : **MCTS**

- On utilise la méthode de Monte Carlo Tree Search pour trouver les configurations gagnantes.

# 06

# IA ALÉATOIRE

Pour cette version de l'IA on procède par un jeu aléatoire

Pour les pions :

**Etape 1 :** Choix d'un pion en aléatoire entre 1 et 5

**Etape 2 :** Vérifier que ce pion peut bouger

**Etape 3 :** Choisir un des déplacements en aléatoire

**Etape 4 :** Réaliser le mouvement

Pour le bobail :

**Etape 1 :** Repérer le bobail

**Etape 2 :** Choisir un mouvement au hasard

**Etape 3 :** Vérifier sa faisabilité

On initialise le temps pour la fonction rand dans la fonction main.

```
CouleurPion** position_bobai_alea (CouleurPion** grille){
    //srand(time(NULL));
    int ligne = (rand()%3)-1;
    int colonne = (rand ()%3)-1;
    int ligne_b;
    int colonne_b;
    for (int i =0; i<5; i++){
        for (int j= 0; j<5; j++){
            if (grille[i][j]==VERT){
                ligne_b=i;
                colonne_b=j;
            }
        }
    }

    if ((ligne_b+ligne)<=4 && (ligne_b+ligne)>=0 && (colonne_b+colonne)<=4 && (colonne_b+colonne)>=0 ){
        if (grille[ligne_b+ligne][colonne_b+colonne]==BLANC){
            grille[ligne_b+ligne][colonne_b+colonne]=VERT;
            grille[ligne_b][colonne_b]=BLANC;
        }
        else{
            position_bobai_alea(grille);
        }
    }
    else {
        position_bobai_alea(grille);
    }
    return grille;
}
```

Exemple pour la fonction  
position\_bobail\_alea

# IA UPDATED

Cette version de l'IA est une **version simplifiée** du Monte Carlo Tree Search. On simule un certain nombre de parties et on **recense le nombre de victoires** pour le choix de la configuration à jouer.

- Etape 1 :** Création d'une liste de tableaux CouleurPions\*\* et initialisation de chaque configuration avec un déplacement aléatoire
- Etape 2 :** Pour chacune des configurations précédentes on lance une centaine de simulation de jeu et on note la somme des victoires
- Etape 3 :** choix de la configuration gagnante selon le max de victoires

## Fonctions utiles



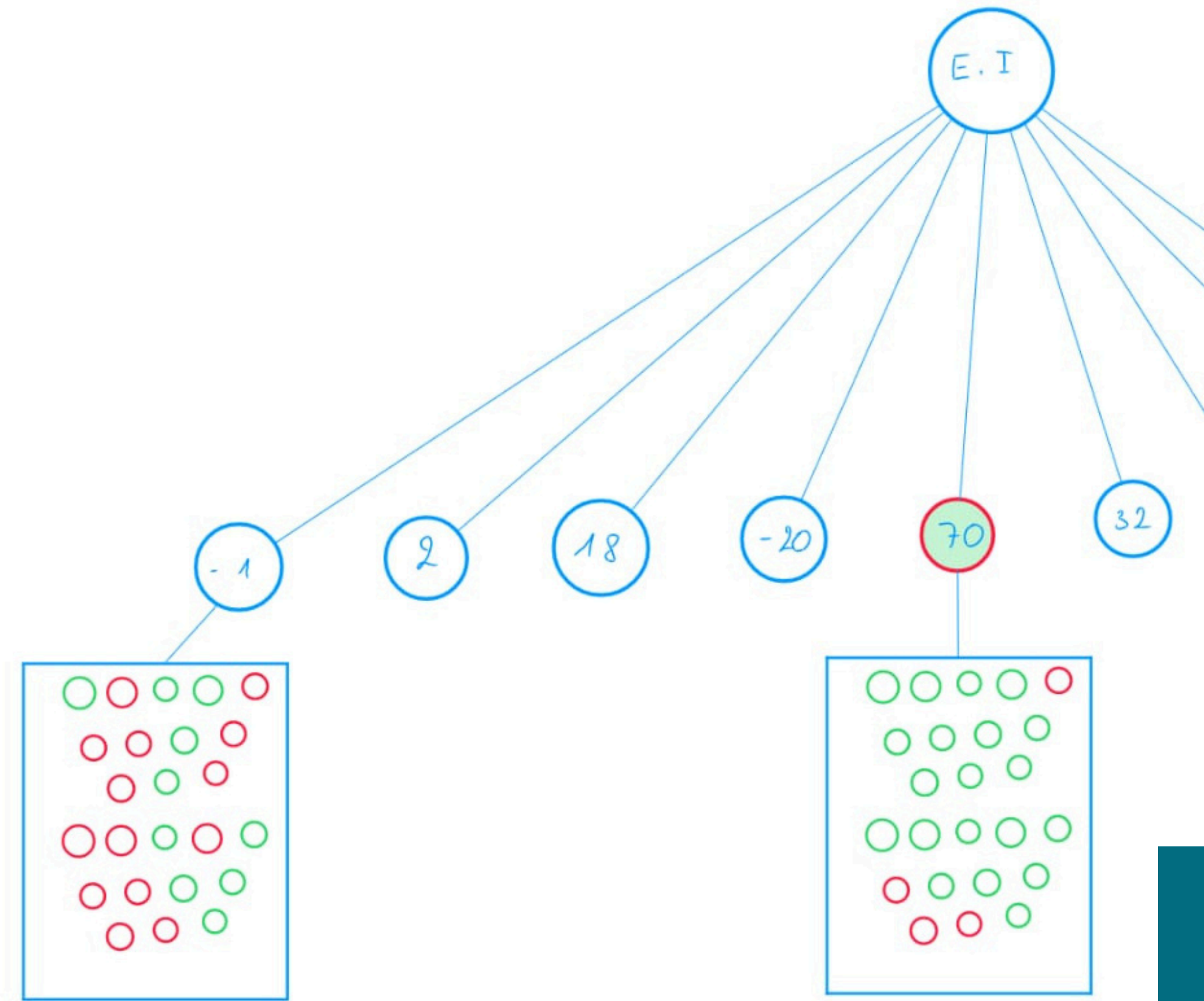
copy\_grille



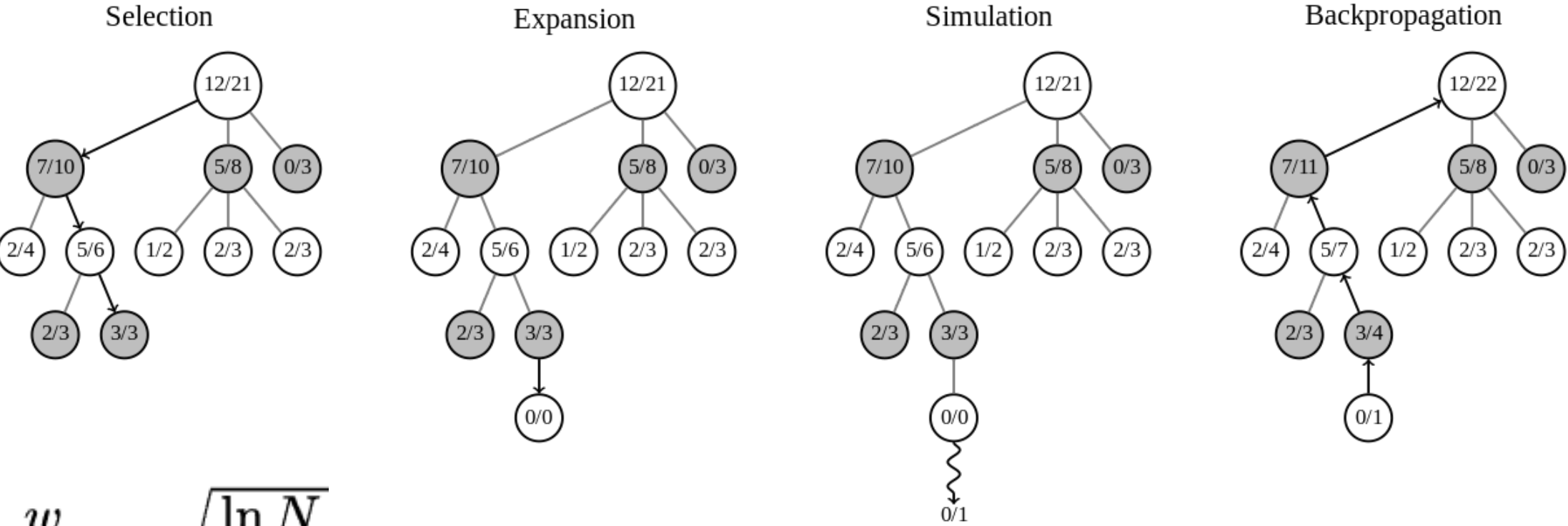
Simu\_jeu



IA\_up




IMPLÉMENTATION DES IA




$$\frac{w}{n} + c\sqrt{\frac{\ln N}{n}}$$

- w est le nombre de parties gagnées marqué dans le successeur i
- n est le nombre de fois où le successeur i a été visité
- N est le nombre de fois où le nœud, père de i, a été visité
- c est le paramètre d'exploration — en théorie égal à  $\sqrt{2}$ , en pratique, choisi expérimentalement.


Fonctions utiles




copy\_grille




Crea\_node




UCB



simu\_jeu



backpropagate



UCB

IA MCTS

Pour notre IA finale on a travaillé avec l’algorithme **Recherche arborescente Monte-Carlo**. On a créé des structures de noeud pour réaliser notre arbre selon un principe de **liste chaînée** pour pouvoir remonter à la configuration gagnante. Les calculs de gain de chaque configuration se font selon la **formule UCB1**

```
typedef struct Noeud {
    CouleurPion** etat;
    struct Noeud* parent;
    struct Noeud* children[MAX_MOVE];
    int simul;
    int victoire;
} Noeud;
```

CONCLUSION

07



## CONCLUSION

---

# CONCLUSION

- GESTION D'ÉQUIPE
- IMPLÉMENTATION D'IA

# PROBLÈMES RENCONTRÉS

- UTILISATION DU GITHUB
- PENSER À TOUTES LES ERREURS
- ÉCRIRE UN CODE CLAIR ET OPTIMISÉ
- COMPRENDRE MCTS ET L'IMPLÉMENTER

**LE JEU EST TROP DUR JE SUIS  
TROP NUL - HIPPOLYTE**

**FOUTU SEGMENTATION FAULT -  
HIPPOLYTE**

”

