# Rapport sur le Projet de Jeu de Stratégie Combinatoire Abstrait en C

## Introduction

Ce rapport présente le développement d'un jeu de stratégie combinatoire abstrait en langage C qu'on a nommé «**Survival Pown** » . Le jeu consiste à déplacer des pions d'un plateau unidimensionnel de cases selon des règles spécifiques jusqu'à ce qu'un joueur remplisse la première case avec le nombre maximal de pions.

## Description du Projet

Le projet est réalisé en langage C et se compose de plusieurs fonctions et procédures pour gérer différents aspects du jeu. Voici une description des principales parties du code :

1. Initialisation du Plateau (newBoard) : La fonction `**newBoard**` initialise le plateau de jeu en attribuant à chaque case un nombre aléatoire de pions, avec une contrainte sur la somme totale des pions pour garantir un déroulement équilibré du jeu.

```c
// Fonction pour initialiser le plateau de jeu
void newBoard(int board[], int n, int p) {
    srand(time(NULL));
    int sum = p;

    for (int i = 0; i < n - 1; i++) {
        int max_pions_case = sum - (n - 1 - i);
        if (max_pions_case > p) {
            max_pions_case = p;
        }
        int random_pions = rand() % (max_pions_case + 1);
        board[i] = random_pions;
        sum -= random_pions;
    }
    board[n - 1] = sum;
}
```

2 Affichage du Plateau (display) : La procédure `**display**` affiche le plateau de jeu sur la console, avec le nombre de pions dans chaque case ainsi que leur numéro.

```c
// Procédure pour afficher le plateau de jeu
void display(int board[], int n) {
    printf("\nPlateau de jeu :\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", board[i]);
    }
    printf("\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", i+1);
    }
    printf("\n");
}
```

3 Sélection d'une Case (selectSquare) : La fonction `**selectSquare**` permet au joueur de sélectionner une case contenant au moins un pion pour effectuer un déplacement.

```c
// Fonction pour sélectionner une case contenant un pion déplacable
int selectSquare(int board[], int n) {
    int selection;
    do {
        printf("Choisissez une case contenant un pion deplacable (1 - %d) : ", n);
        scanf("%d", &selection);
        selection--; // Ajustement pour l'index 0-based
    } while (selection < 0 || selection >= n || !possibleSquare(board, selection));
    return selection;
}
```

4 Vérification de la Destination (possibleDestination) : La fonction `**possibleDestination**` vérifie si une case est une destination valide pour déplacer un pion depuis une case sélectionnée.

```c
// Fonction pour vérifier si une case contient un pion déplacable
bool possibleSquare(int board[], int i) {
    return (board[i] > 0);
}
```

5 Sélection de la Destination (selectDestination) : La fonction `**selectDestination**` permet au joueur de sélectionner une destination valide pour déplacer un pion.

```
// Fonction pour vérifier si une case est une destination valide pour un pion donné
bool possibleDestination(int board[], int i, int j) {
    return (j < i && (board[j] == 0 || board[j] > 0));
}
```

6 Déplacement d'un Pion (move) : La procédure `**move**` réalise le déplacement effectif d'un pion d'une case sélectionnée vers une destination choisie.

```
// Procédure pour déplacer un pion
void move(int board[], int i, int j) {
    board[j] += 1;
    board[i] -= 1;
}
```

7 Vérification de la Victoire (win) : La fonction `**win**` vérifie si le joueur a gagné en remplissant la première case avec le nombre maximal de pions.

```
// Fonction pour vérifier si le joueur a gagné
bool win(int board[], int n, int p) {
    return (board[0] == p);
}
```

8 Vérification de la Défaite (lose) : La fonction `**lose**` vérifie si le joueur a perdu en ne pouvant plus déplacer de pions.

```
// Fonction pour vérifier si le joueur a perdu
bool lose(int board[], int n) {
    for (int i = 0; i < n; i++) {
        if (board[i] > 0) {
            return false;
        }
    }
    return true;
}
```

9 Jeu Principal (jeu) : La procédure `**jeu**` orchestre le déroulement complet du jeu en utilisant les fonctions et procédures précédentes. Elle permet à deux joueurs de jouer alternativement jusqu'à ce qu'un joueur gagne ou que la partie se termine par une égalité.

```
// Procédure pour jouer une partie complète
void jeu(int n, int p) {
    int board[n];
    newBoard(board, n, p);
    display(board, n);
    int currentPlayer = 1;
    while (!lose(board, n)) {
        printf("\nJoueur %d :\n", currentPlayer);
        int square = selectSquare(board, n);
        int destination = selectDestination(board, square);
        move(board, square, destination);
        display(board, n);
        if (win(board, n, p)) {
            printf("\nLe joueur %d a gagne !\n", currentPlayer);
            return;
        }
        currentPlayer = (currentPlayer == 1) ? 2 : 1;
    }
    printf("\nLe joueur %d a perdu.\n", currentPlayer);
}
```

## Explication du Code

Le code est structuré de manière modulaire, avec des fonctions et des procédures distinctes pour chaque fonctionnalité du jeu. Cela permet une meilleure lisibilité, maintenance et évolutivité du code. Voici les principales étapes du déroulement du jeu :

- Initialisation du Jeu : Le jeu commence par demander à l'utilisateur le nombre de cases et le nombre maximal de pions par case. Ensuite, le plateau est initialisé avec la fonction `**newBoard**` et affiché à l'écran.

- Tour des Joueurs : Les joueurs jouent à tour de rôle en sélectionnant une case contenant un pion déplaçable, puis en choisissant une destination valide pour le déplacement de ce pion. Le plateau est mis à jour après chaque déplacement et réaffiché à l'écran.

- Fin du Jeu : Le jeu se termine lorsque l'un des joueurs remplit la première case avec le nombre maximal de pions, ce qui déclenche la victoire de ce joueur. Si aucun joueur ne peut déplacer de pions, la partie se termine par une égalité.

## Remerciements

Je tiens à exprimer ma sincère reconnaissance envers toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce projet. Leur soutien et leur collaboration ont été précieux tout au long du processus de développement.

Je remercie également toutes les personnes qui ont partagé leurs connaissances et leur expertise, et qui ont apporté des conseils et des suggestions constructives.

Enfin, je tiens à exprimer ma gratitude envers mes proches pour leur encouragement constant et leur soutien inconditionnel tout au long de ce projet.

## Conclusion

Ce projet a permis de développer un jeu de stratégie combinatoire abstrait en langage C, en respectant les règles spécifiées dans le cahier des charges. Le code est bien structuré, modulaire et offre une expérience de jeu interactive et engageante pour deux joueurs. Il peut être amélioré avec des fonctionnalités supplémentaires telles que la gestion des scores, des niveaux de difficulté ou une interface graphique pour une meilleure expérience utilisateur.