Informatique
Hackathon 1

TP P

v2025-09-24

IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy

Sommaire

1 Projet 1 — Le 421 simplifié

Règles du jeu

- Le jeu se joue à deux joueurs.
- Chaque joueur dispose de 30 points au début de la partie.
- Le premier joueur lance trois dés.
- Il peut ensuite relancer un ou plusieurs dés, dans la limite de trois lancers au total.
- Il peut s'arrêter avant le troisième lancer s'il le souhaite.
- Le second joueur joue ensuite de la même manière et doit effectuer exactement le même nombre de lancers que le premier.
- À la fin de la manche, on compare les combinaisons. Le vainqueur prend à son adversaire un nombre de points dépendant de sa combinaison.
- Le vainqueur d'une manche commence la manche suivante.

Combinaisons gagnantes

Voici les combinaisons possibles, classées de la meilleure à la moins bonne :

Combinaison	Description	Points
4–2–1	Spéciale	**21**
Triplette	Trois dés identiques	Valeur du dé $\times 2$
Paire	Deux dés identiques (le troisième départage)	Valeur du dé de la paire
Aucune	Le plus grand nombre formé l'emporte	1

Cahiers des charges

Cahier des charges 1 : Fonctionnement normal - Score manuel

Le programme
- Explique les règles
- Permet aux joueurs de jouer au jeu, en gérant les lancers et relances
- Seul le score de chaque manche n'est pas calculé automatiquement :
  - ◇ Le programme demande le vainqueur
  - ◇ Le programme demande le nombre de points à transférer

Cahier des charges 2 : Score automatique

Calcul du score automatique.

Cahier des charges 3 : L'ordinateur joue

Mode solo : jouer contre l'ordinateur.

Cahier des charges 4 : Idées d'améliorations

- Afficher les dés sous forme de carrés
- Ajouter les combinaisons manquantes du vrai jeu 421
- Autres ?

Aide pour le projet

```c
int main(void) {
int tab_joueurs_points[2]; //Déclaration du tableau de point des 2 joueurs
int tab_jeu[6]; //Déclaration du tableau de jeu. Les 3 chiffres du tirage du joueur 1 seront rangés dans
     les 3 premières cases de ce tableau, les 3 chiffres du joueur 2 seront rangés dans les 3 suivantes.
bool joueur = 0;
int iBcl = 0 ;
char lettre = 0;

srand(time(NULL)); // Initialise le générateur aléatoire. Nécessite #include <time.h>

do{
    printf("Joueur %d joue :\n", joueur);

    //Boucle for qui lance 3 dés et mémorise dans le tableau de jeu et affiche :
    for(iBcl = 0;iBcl<3;iBcl++){
        tab_jeu[l_joueur * 3+iBcl] = rand(); //Utilisez % !!!
        printf("Dé %d de joueur %d = %d\n", iBcl+1, joueur, tab_jeu[joueur*3+iBcl]);
    }

    lettre = get_char("Appuyez sur une touche pour passer au joueur suivant. q pour sortir.\n");
    joueur = !joueur; //Pour passer au 2ème joueur
}while(lettre != 'q');

return 0;
}
```

2 Projet 2 — Le compte est bon

Objectif du jeu

Obtenir une **cible** comprise entre 100 et 999 en combinant **six nombres tirés** dans une liste donnée, à l'aide des opérations $+, -, \times, /$ (division exacte uniquement).

Règles du jeu

- Six nombres sont tirés au hasard dans la liste officielle : deux occurrences de 1 à 10, puis 25, 50, 75, 100.
- Une **cible** aléatoire entre 100 et 999 est annoncée.
- Chaque nombre peut être utilisé **au plus une fois**.
- Les opérations autorisées sont $+, -, \times, /$ et les divisions doivent être **entières**.
- Une manche dure **60 secondes** (vous pouvez réduire à 3 s pour les tests).

Cahiers des charges

Cahier des charges 1 : Jeu simple

- Tirage des **6 nombres** et d'une **cible** puis affichage.
- A la fin du temps, le joueur **annonce** s'il a réussi ou non.
- S'il a réussi, il gagne un point. Sinon, il perd un point (il ne peut pas avoir de points négatifs).
- Le joueur gagne s'il atteint 3 points.

Aide : utilisez `srand(time(NULL))`, `rand()` et `%` pour borner les tirages ; un chronométrage simple peut se faire avec `time(NULL)` et une boucle d'attente.

Cahier des charges 2 : Tirage sans remise

- Tirage **sans remise** à partir du tableau de 24 valeurs (respect des probabilités).

On considère que les nombres sont tirés à partir d'un panier dans lequel les balles ne sont pas replacées après tirage. Ainsi, si, par exemple, un 5 est tiré, il n'en reste plus qu'un au lieu de deux dans le panier. En programmation, cela revient à tirer au sort dans le tableau et à ne plus pouvoir tirer la case au sort. *Proposition de technique :* « **swap avec la dernière case** » : lorsqu'une case est choisie, copiez la dernière case à sa place et réduisez la borne de tirage ($24 \to 23 \to \ldots$).

Cahier des charges 3 : Saisie des opérations

- L'ordinateur affiche les nombre disponibles et le résultat à obtenir
- Le joueur saisi les opérations au clavier
- Après chaque opération, l'ordinateur retire les nombres déjà utilisés et affiche les nouveau nombres disponibles (résultats de l'opération). Par exemple :
  1. Cible : 645, avec 50, 6, 7, 1, 5, 2
  2. Le joueur saisit l'opération 6 + 7
  3. Cible : 645, avec 50, 13, 0, 1, 5, 2
  4. etc.
- Si la cible est atteinte, l'ordinateur affiche **Le compte est bon**

Cahier des charges 4 : Améliorations possibles

- Affichage du compte à rebours
- ...

Squelette de départ

```c
int main(void) {
    // Tableau officiel (2x pour 1..10, puis 25, 50, 75, 100)
    int l_tab_nombres[24] = {
        1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,25,50,75,100
    };

    // Les 6 nombres tirés
    int l_tab_jeu[6] = {0};

    srand(time(NULL)); // Initialise le générateur aléatoire (une seule fois)

    // --- Tirage naf (Niv.1) :  améliorer  sans remise  au Niv.2 ---
    for (int i = 0; i < 6; i++) {
        int idx = rand(); // utilisez % pour borner
        l_tab_jeu[i] = l_tab_nombres[idx];
        printf("Tirage %d = %d\n", i + 1, l_tab_jeu[i]);
    }

    // --- Cible dans [100..999] (corriger la borne) ---
    int cible = rand(); // utilisez % pour borner dans [100..999]
    printf("Cible %d\n", cible);

    // --- Chrono de 60 s (tolérance 3 s en test) ---
    // Astuce : l_start = time(NULL); puis boucle jusqu'à (time(NULL)-l_start) >= 60

    return 0;
}
```

3 Projet 3 — Bataille navale

Objectif du jeu

Couler les bateaux cachés dans une **mer 6×6** en proposant des **coordonnées** de tir. Le jeu se joue **joueur vs processeur**.

Règles du jeu

- La mer est une grille 6×6.
- Deux bateaux : **Bateau n°1** (2 cases) et **Bateau n°2** (3 cases).
- Les bateaux sont placés **aléatoirement**, **horizontalement ou verticalement**, sans chevauchement et entièrement dans la grille.
- Le joueur propose des coordonnées. **Touché** si la case contient un bateau, **Coulé** lorsque toutes les cases d'un même bateau sont touchées.
- **Victoire** quand les deux bateaux sont coulés ; afficher le **nombre de tentatives** et le **temps de jeu**.

Codage conseillé des cases (pour la logique interne)

Valeur	Signification
0	Case vide
1	Bateau n°1, non touchée
10	Bateau n°1, touchée
2	Bateau n°2, non touchée
20	Bateau n°2, touchée

Cahiers des charges

Cahier des charges 1 : Grille, placement et tirs

- Le joueur joue contre l'ordinateur et tente de couler ses bateaux.
- La mer doit être affichée après chaque tir, avec les cases déjà tirées marquées (Tirées vides / Touchées).

Cahier des charges 2 : Deux joueurs

Un joueur place les bateaux, l'autre tente de les couler. Alterner les rôles.

Cahier des charges 3 : Deux joueurs

Le vrai jeu. Chaque joueur place ses bateaux en début de partie (sans les montrer à l'autre). Les joueurs jouent à tour de rôle, chacun tentant de couler les bateaux adverses.

Squelette de départ

```c
int main(void){
    int l_tab_mer[6][6]; //Déclaration de la mer
    int iBclX, iBclY;

//Affichage de la première ligne de la grille (numéro de colonne) :
```

```
printf(" ");

for(iBclY=0;iBclY<6;iBclY++){
    printf("%i ",iBclY+1);
}

//Initialisation et affichage de la grille :

printf("\n");
for(iBclY=0;iBclY<6;iBclY++){ //Boucle des abscisses
    //Affichage de la lettre de la ligne :
    printf("%c ", iBclY+65);

    for(iBclX=0;iBclX<6;iBclX++){ //Boucle des ordonnées
    l_tab_mer[iBclY][iBclX] = 0; //Initialisation à 0
    printf("%i ",l_tab_mer[iBclY][iBclX]); //Affichage
    }

    printf("\n");
}

return 0;
}
```

4 Projet 4 — Jeu Simon

Objectif du jeu

Tester la mémoire des joueurs en construisant une **séquence de caractères** de plus en plus longue. À chaque tour, le joueur courant doit reproduire la séquence précédente puis **ajouter un nouveau caractère**.

Règles du jeu

- Le jeu se joue à deux personnes qui jouent en alternance, ou contre l'ordinateur.
- Le premier joueur saisit un **caractère minuscule** affiché à l'écran.
- Le second joueur doit alors saisir la même séquence **plus** un nouveau caractère. **Seul le dernier caractère saisi est affiché**. Si la séquence est incorrecte, il perd et le jeu s'arrête.
- Si la séquence du second joueur est juste, on repasse la main au premier joueur, et ainsi de suite : à chaque tour, on **rejoue la séquence complète** puis on ajoute un caractère.
- Deux offres de jeu : **Jeu 1** (2 joueurs physiques) ; **Jeu 2** (contre l'ordinateur, on affiche le **nombre de coups** en fin de partie).

Cahier des charges

Cahier des charges 1 : Séquence à deux joueurs

- Gérer la construction de la séquence **commune** (longueur max 50) et l'alternance des joueurs.
- À chaque tour : reproduire la séquence précédente, puis saisir **un nouveau caractère** (afficher uniquement le **dernier** caractère saisi).
- En cas d'erreur de séquence, **annoncer le perdant** et arrêter la partie.

Aide : une **seule boucle** pour les deux joueurs (même code) avec une variable de joueur (`0/1`) ; afficher la séquence courante en parcourant le tableau de 0 à `nbrCoup`.

Cahier des charges 2 : Mode contre l'ordinateur

- L'ordinateur joue en alternance avec le joueur humain et **ajoute** un caractère à la séquence.
- En fin de partie, afficher le **nombre de coups** joués.
- Prévoir **plusieurs niveaux de difficulté** (fréquence d'erreurs de l'IA).

Aide : générer le caractère de l'ordinateur aléatoirement, et introduire une « erreur » selon une probabilité paramétrable.

Cahier des charges 3 : Robustesse et présentation

- Valider les saisies (lettre minuscule), gérer la **taille max** de 50 caractères et les **messages** de statut.
- Proposer un **menu** : Jeu 1 (2 joueurs) / Jeu 2 (contre l'ordinateur).
- Soigner l'affichage (Rendre le jeu le plus beau possible)

Aide : stocker la séquence correcte et la séquence saisie dans `char tab[50]` ; `time(NULL)` pour initialiser l'aléatoire.

Squelette de départ

```c
int main(void) {
    char l_tab_jeuCorrect[50]; // Déclaration du tableau de jeu
    char l_tab_joueur[50];     // Déclaration du tableau du joueur pour la saisie
    int l_nbrCoup = 0, l_joueur = 0, iBcl = 0;

    srand(time(NULL)); // Initialise le générateur aléatoire. Nécessite #include <time.h>
```

```
    // On impose une seule boucle pour les 2 joueurs (même code) :
    do {
        printf("Joueur %d joue\n", l_joueur);

        l_tab_jeuCorrect[l_nbrCoup] = get_char("Quelle lettre ?");

        printf("Le tableau est : \n");
        for (iBcl = 0; iBcl <= l_nbrCoup; iBcl++) {
            printf("%c ", l_tab_jeuCorrect[iBcl]);
        }
        printf("\n\n");

        l_joueur = !l_joueur; // Pour passer au 2ème joueur
        l_nbrCoup++;

    } while (1); // Boucle infinie

    return 0;
}
```

5 Projet 5 — Jeu des couleurs

Objectif du jeu

L'ordinateur tire une **suite de 10 couleurs** parmi *noir, vert, rouge, bleu*. Les couleurs s'affichent brièvement, puis disparaissent. Le joueur doit **reproduire la séquence** dans le bon ordre (codage numérique). Le score et la durée sont affichés en fin de partie. Un mode 2 joueurs est ensuite proposé.

Règles du jeu

- L'ordinateur choisit 10 couleurs au hasard parmi {noir, vert, rouge, bleu}.
- Les 10 carrés de couleurs s'affichent en ligne pendant 5 s, puis l'écran est effacé.
- Le joueur saisit la séquence en codes : `0` pour noir, `1` pour bleu, `2` pour vert, `3` pour rouge.
- À chaque saisie correcte, **+1 point**. À chaque erreur, **-1 point**.
- En fin de partie : Afficher le **score**.

Cahier des charges

Cahier des charges 1 : Mode 1 joueur simplifié

- Affichage de la séquence, puis effacement de l'écran
- Le joueur tente de reformer la suite
- Affichage du score en fin de partie.

Aide : utiliser `srand(time(NULL))`, `rand()%4`, `sleep(5)`, `system("clear")` ; ANSI `\033[40m..\033[0m` pour les carrés.

Cahier des charges 2 : Mode 1 joueur complet

- A la fin de la partie, la séquence originale et la séquence du joueur sont affichées pour comparaison
- La durée et le score sont également affichés

Cahier des charges 3 : Mode 2 joueurs et bilan complet

- Chaque joueur saisit sa propre **séquence** ; comparer les scores et **annoncer le vainqueur**.
- Afficher les **trois séquences** : tirage, joueur 1, joueur 2, ainsi que leurs scores et durées.
- Soigner la **présentation** (espaces, retours à la ligne, lisibilité des carrés).

Aide : réutiliser l'affichage des carrés et une fonction de calcul de score pour éviter la duplication.

Squelette de départ

```c
void printSquareColor(int x_code);

//prototype de la fonction printSquareColor()
int main(void) {

    srand(time(NULL));  // Initialise le générateur aléatoire. Nécessite #include <time.h>
    printf("\n");

    //Boucle qui affiche 10 carrés de couleurs aléatoires
    for(int iBcl = 0;iBcl<10;iBcl++){
        printSquareColor(rand()%4);
        printf(" ");

    }
```

```c

    printf("\n");
    sleep(5); // Attente de 5 secondes bloquante
    system("clear"); // Efface le terminal
    return 0;

}

//fonction printSquareColor() qui affiche un carré. Le paramètre d'entrée indique le code de  la couleur du carré souhaitée. Pas de retour.
void printSquareColor(int x_code){

    switch(x_code){

        case 0 :
//0 pour afficher un carré noir
            printf("\033[40m  \033[0m");
        break;

        case 1 :
//1 pour afficher un carré bleu
            printf("\033[41m  \033[0m");

        break;

        case 2 :
//2 pour afficher un carré vert
            printf("\033[42m  \033[0m");
        break;

        case 3 :
//3 pour afficher un carré rouge
            printf("\033[44m  \033[0m");
        break;

    }

}
```

6 Projet 6 — Poker

Objectif du jeu

Construire un jeu simplifié de Poker à 5 cartes pour un joueur, reconnaître les combinaisons demandées et calculer un score. Étendre ensuite à deux joueurs.

Règles du jeu

- On joue avec un **jeu de 32 cartes** (valeurs 7,8,9,10, V, D, R, As) et 4 couleurs.
- Le joueur reçoit **5 cartes** tirées au hasard **sans remise**. Il peut remplacer une ou plusieurs cartes.
- Combinaisons gagnantes : **paire** (+1), **brelan** (+2), **full** (+3), **carré** (+5).
- Afficher la main et la combinaison gagnante éventuelle. En version deux joueurs, **la meilleure main gagne**.

Cahier des charges

Cahier des charges 1 : Jeu de base

- Tirer **5 cartes sans remise** depuis le jeu de 32 cartes, les afficher (couleur et valeur).
- Permettre au joueur d'indiquer les cartes à remplacer (ex. un entier `134` pour 1,3,4), puis **retirer** sans remise.
- Afficher la main finale.

Aide : tirage sans remise par « échange avec la dernière case » (réduire la borne).

Cahier des charges 2 : Score automatique

- Détecter automatiquement : **paire**, **brelan**, **full**, **carré** et **calculer le score**.
- Afficher un **récapitulatif** clair (main, combinaison, score).
- Vérifier la **saisie** des cartes à remplacer.

Aide : ranger les valeurs des 5 cartes, compter les occurrences.

Cahier des charges 3 : Mode deux joueurs

- Mode **deux joueurs** : chacun joue sa main, comparaison des combinaisons et **annonce du vainqueur**.
- Présentation soignée, possibilité de **rejouer** plusieurs manches.

Squelette de départ

```c
int main(void) {
    // 72 = 'H' pour Heart (Coeur), 80 = 'P' pour Pique, 84 pour 'T' Trefle et 67 pour 'C' Carreau
    int l_tab_jeu32[32] = {7201,7207,7208,7209,7210,7211,7212,7213,
     8001,8007,8008,8009,8010,8011,8012,8013, 8401,8407,8408,8409,8410,8411,8412,8413,
     6701,6707,6708,6709,6710,6711,6712,6713};
    int iBcl;

    for(iBcl=0;iBcl<32;iBcl++){
        printf("%c ", l_tab_jeu32[iBcl]/100);
        printf("%i \n", l_tab_jeu32[iBcl]%100);
    }
    return 0;
}
```

7 Projet 7 — Le morpion

Objectif du jeu

Réaliser un morpion (3×3) jouable à deux joueurs au clavier, avec affichage ASCII et détection de victoire/égalité.

Règles du jeu

- La grille est un tableau de **9 cases** : `0` = vide, `1` = 'X' (J1), `2` = 'O' (J2).
- Les joueurs jouent à tour de rôle. **Même code** pour les deux joueurs (variable joueur).
- Un joueur gagne s'il aligne 3 symboles en ligne, colonne ou diagonale. Sinon, partie nulle.

Cahier des charges

Cahier des charges 1 : 1

- Boucle principale avec alternance J1/J2, **saisie** d'une case et **affichage** de la grille.
- Affecter 1 ou 2 selon le joueur, sans validation initiale.
- Afficher la grille après chaque coup.

Cahier des charges 2 : 2

- **Valider** la saisie (case existante et libre), sinon rejouer.
- **Détecter** la victoire (lignes, colonnes, diagonales) et l'égalité.
- Présenter un **récapitulatif** en fin de partie.

Cahier des charges 3 : 3

- Ajouter un **mode IA** basique (ne pas perdre) ou des options (rejouer).
- Structurer le code (fonctions d'affichage et de test).

Squelette de départ

```c
int main(void) {

    int l_tab_jeu[9] = {1,0,0,0,2,0,1,0,2};

    int iBcl1 = 0, iBcl2 = 0, l_joueur = 0, l_cpt = 0;

    for(iBcl2 = 0 ; iBcl2 <3 ; iBcl2++){
        printf("\n+---+---+---+\n|");
        for(iBcl1 = 6-3*iBcl2 ; iBcl1<9-3*iBcl2;iBcl1++){
            printf(" %c |",(l_tab_jeu[iBcl1]==1)*'X' + (l_tab_jeu[iBcl1]==2)*'O' + (l_tab_jeu[iBcl1]==0)
 *32);
        }
    }
    printf("\n+---+---+---+\n");

    do{
        l_joueur = (l_cpt%2) + 1 ;
        printf("\n\nJoueur %i joue", l_joueur);
        l_cpt++;
    }while(1);

```

```
    return 0;
}
```

8 Projet 8 — Puissance 4

Objectif du jeu

Programmer un Puissance 4 sur une grille 6×6 en ASCII, jouable à deux, avec détection de 4 alignés.

Règles du jeu

- Une **grille 6×6** commune aux deux joueurs (1 et 2).
- À son tour, un joueur choisit une **colonne** ; le pion tombe dans la **case libre la plus basse**.
- Afficher la grille après chaque coup. Le premier à aligner 4 pions gagne (—, | ou /).

Cahier des charges

Cahier des charges 1 : 1

- Initialiser la grille à 0 et l'afficher.
- Boucle principale avec alternance J1/J2 et **choix d'une colonne** (sans validation initiale).
- Placer le pion en bas de la colonne, réafficher.

Cahier des charges 2 : 2

- **Valider** la colonne (existence et place libre), sinon rejouer.
- Détecter **4 alignés** et l'égalité.
- Récapitulatif de fin.

Cahier des charges 3 : 3

- Mode **joueur vs ordinateur** simple.
- Rejouabilité et présentation.
- Factoriser en fonctions (affichage, pose, test alignements).

Squelette de départ

```c
int main(void) {

    int l_tab_grille[6][6];
    int iBclX = 0, iBclY = 0, l_joueur = 0, l_cpt = 0;

    for(iBclX=0;iBclX<6;iBclX++){
        for(iBclY=0;iBclY<6;iBclY++){
            l_tab_grille[iBclX][iBclY] = 0;
            printf("%i ",l_tab_grille[iBclX][iBclY]);
        }
        printf("\n");
    }

    do{
        l_joueur = (l_cpt%2) + 1 ;
        printf("\n\nJoueur %i joue", l_joueur);
        l_cpt++;
    }while(1);

    return 0;
}
```

9 Projet 9 — Le bandit manchot

Objectif du jeu

Simuler un bandit manchot à 3 rouleaux et gérer le capital du joueur selon les combinaisons obtenues.

Règles du jeu

- 3 **rouleaux** de 11 figures (tableau 2D). Codes : `7, B, L, C` (sept, barre, citron, cerise).
- Le joueur commence avec **100 points** et mise 1 ou 3 points.
- Appui sur `J` : tirage aléatoire d'une figure par rouleau ; affichage en ligne des trois codes.
- Gagne/perd selon les combinaisons définies ; le capital est mis à jour ; le joueur peut continuer (`O/N`).

Cahier des charges

Cahier des charges 1 : 1

- Initialiser les **3 rouleaux** (tableau 3×11) et afficher leur contenu pour vérification.
- Tirer et afficher **une figure par rouleau** (codes).
- Mettre à jour le capital selon résultat simple (ex. égalité des 3).

Cahier des charges 2 : 2

- Implémenter les **mises** (1 ou 3), détailler les combinaisons gagnantes/perdantes.
- Autoriser de **conserver** un ou plusieurs rouleaux (ex. saisie 23) avec pénalité de points.
- Récapitulatif clair de fin.

Cahier des charges 3 : 3

- Présentation et rejouabilité ; options de test (rouleaux biaisés).
- Structuration en fonctions (tirage, calcul gain, affichage).
- Robustesse des saisies.

Squelette de départ

```c
int main(void) {

    char l_tab_rouleau[3][11] =     {{'L','7','C','B','L','C','L','L','C','L','B'},
                                     {'L','C','7','L','C','L','B','3','L','7','C'},
                                     {'L','B','L','C','L','7','C','L','B','C','L'}};

    int iBclX, iBclY;

    for(iBclX=0;iBclX<3;iBclX++){
        for(iBclY=0;iBclY<11;iBclY++){
            printf("%c ",l_tab_rouleau[iBclX][iBclY]);
        }
        printf("\n");
    }

    return 0;
}
```

10 Projet 10 — Le pendu

Objectif du jeu

Faire deviner un mot lettre par lettre avec 7 erreurs autorisées, en affichant la progression.

Règles du jeu

- Un utilisateur saisit un **mot** (minuscule, sans accents ni espaces).
- Un joueur propose des lettres ; le programme affiche le mot avec _ pour les lettres non trouvées.
- À chaque lettre absente, **+1 erreur**. La partie s'arrête à 7 erreurs ou si le mot est trouvé.
- Variante 2 joueurs : inversion des rôles, 3 manches.

Cahier des charges

Cahier des charges 1 : 1

- Saisir le mot **secret**, l'afficher **caractère par caractère** pour vérification, puis masquer.
- Boucle de jeu : saisie d'une lettre, mise à jour de l'affichage, **comptage des erreurs**.
- Condition de victoire/défaite.

Cahier des charges 2 : 2

- Validation des saisies, gestion des lettres déjà proposées.
- Affichage du temps de jeu.

Cahier des charges 3 : 3

- Présentation soignée -> Dessin ?
- Mode 2 joueurs avec alternance des rôles sur 3 manches.
- Option : dictionnaire de mots.

Squelette de départ

```
int main(void)
{
    string l_mot;
    int iBcl = 0;

    l_mot = get_string("Entrer le mot \n");
    do{
        printf("%c\n",l_mot[iBcl]);
        iBcl++;
    }while(l_mot[iBcl] != 0);
}
```

11 Projet 11 — Mastermind – Suite de couleurs

Objectif du jeu

Deviner une suite cachée de 4 couleurs codées (0..3) avec indication des bien/mal placées.

Règles du jeu

- Le programme tire au hasard une **suite de 4 couleurs** parmi Rouge, Vert, Bleu, Noir (**avec répétitions possibles**).
- Codage : `0`=Noir, `1`=Rouge, `2`=Vert, `3`=Bleu.
- Le joueur saisit un **entier à 4 chiffres** ; le programme le **décompose** en tableau.
- Le programme indique **nombre de bien placées** et **nombre de mal placées**. Score = tentatives × temps / 10.

Cahier des charges

Cahier des charges 1 : 1

- Tirer la suite et **mémoriser** le code (tableau 4 cases) ; **ne pas l'afficher**.
- Le joueur saisit la proposition (entier 4 chiffres), **décomposer** en tableau, comparer et afficher les compteurs.
- Boucler jusqu'à la bonne réponse.

Cahier des charges 2 : 2

- Gérer les répétitions correctement (bien/mal placées).
- Ajouter **chrono** et calcul du score ; récapitulatif.
- Valider la saisie.

Cahier des charges 3 : 3

- Mode **2 joueurs** en alternance ; afficher les tentatives de chacun.
- Présentation (affichage de carrés ANSI) et rejouabilité.
- Factoriser en fonctions.

Squelette de départ

```c
void printSquareColor(int x_code);

int main(void) {

    int iBcl;
    printf("\n");

    for(iBcl = 1;iBcl<=4;iBcl++){
        printSquareColor(iBcl);
        printf(" ");
    }
    return 0;
}

void printSquareColor(int x_code){

    switch(x_code){
```

```
        case 0 :
            printf("\033[40m  \033[0m");
        break;

        case 1 :
            printf("\033[41m  \033[0m");
        break;

        case 2 :
            printf("\033[42m  \033[0m");
        break;

        case 3 :
            printf("\033[44m  \033[0m");
        break;
    }
}
```

12 Projet 12 — Black Jack adapté

Objectif du jeu

Adapter un Black Jack simplifié avec un jeu de 32 cartes, tirage sans remise, calcul de la main et du gagnant.

Règles du jeu

- Jeu de **32 cartes** encodées `CCVV` (couleur×100 + valeur). Couleurs : 72=’H’ (coeur), 80=’P’ (pique), 84=’T’ (trèfle), 67=’C’ (carreau).
- Valeurs des cartes : 1 (As),2,3,4,5,6,7,8,9,10,11 (V), 12 (D), 13 (R).
- Un joueur reçoit **2 cartes sans remise** ;
- Il peut demander des cartes supplémentaires, autant qu’il le souhaite, mais si son score dépasse 21, son score devient 0 et son tour se termine.
- Le programme doit alors tirer le même nombre de cartes que le joueur et le meilleur score gagne.

Cahier des charges

Cahier des charges 1 : 1

- Affichage des cartes tirées et propositions de tirer de nouvelles cartes.
- Afficher la main, la valeur et **la combinaison gagnante éventuelle**.

Cahier des charges 2 : 2

- En cas de match nul, on ajoute une carte aux deux joueurs

Cahier des charges 3 : 3

- Mode **2 joueurs** ; comparaison et annonce du gagnant.
- Rejouabilité et présentation.
- Factoriser en fonctions.

Squelette de départ

```c
int main(void) {

    int l_tab_jeu32[32] = {7201,7207,7208,7209,7210,7211,7212,7213,
     8001,8007,8008,8009,8010,8011,8012,8013, 8401,8407,8408,8409,8410,8411,8412,8413,
     6701,6707,6708,6709,6710,6711,6712,6713};
    int iBcl;

    for(iBcl=0;iBcl<32;iBcl++){
        printf("%c ", l_tab_jeu32[iBcl]/100);
        printf("%i \n", l_tab_jeu32[iBcl]%100);
    }

    return 0;
}
```

13 Projet 13 — Penalty !

Objectif du jeu

Simuler une séance de tirs au but avec choix du tireur et du gardien, comptage des buts et affichage du score.

Règles du jeu

- Deux adversaires s'affrontent au **tir au but**. À chaque tir, le tireur choisit une zone, le gardien choisit une zone.
- But si les choix ne coïncident pas ; sinon arrêt. Après une série, on affiche le **score** et le vainqueur.
- Variantes possibles (meilleure des 5, mort subite, difficulté).

Cahier des charges

Cahier des charges 1 : 1

- Version **basique jouable** : saisies tireur/gardien, détection but/arrêt, **compteur de buts** et fin de série.
- Affichage du **score** et du vainqueur.
- Rejouer une série.

Cahier des charges 2 : 2

- Validation des saisies, modes de série (5 tirs, mort subite).
- Ajout d'une **IA** simple pour le gardien.
- Récapitulatif de fin.

Cahier des charges 3 : 3

- Présentation et rejouabilité, statistiques.
- Facteur aléatoire (puissance tir, précision) optionnel.
- Factoriser en fonctions.

14 Projet 14 — Codenames

Objectif du jeu

Mettre en place une version console de *Codenames* permettant de jouer à deux équipes avec un plateau 5×5, un maître-espion par équipe et une gestion claire des tours, révélations et conditions de fin.

Règles du jeu

- **Constitution du plateau** : 25 cartes (mots) sont placées en grille 5×5. Sous ces cartes se cachent des rôles : cartes **équipe Rouge**, cartes **équipe Bleue**, **neutres** et une carte **Assassin**. Une équipe commence (elle a une carte de plus).
- **Rôles** : chaque équipe a un **maître-espion** (voit la carte-clé indiquant la couleur de chaque mot) et des **agents** (devinent).
- **Tour de jeu** : le maître-espion donne **un seul indice** (un mot) suivi d'un **nombre** indiquant combien de cartes de sa couleur sont visées. Les agents discutent et **désignent des cartes une par une**. Chaque désignation révèle la carte.
- **Révélations** : si la carte est de la couleur de l'équipe, elle reste au tour et peut continuer jusqu'à nombre + 1 maximum. Si elle est **neutre**, le tour s'arrête. Si elle est de la **couleur adverse**, le tour s'arrête et l'autre équipe marque la carte. Si c'est l'**Assassin**, l'équipe active **perd immédiatement**.
- **Fin de partie** : dès qu'une équipe a révélé toutes ses cartes, elle **gagne**. Si l'Assassin est révélé, l'autre équipe **gagne** aussitôt.

Cahier des charges

Cahier des charges 1 : 1

- Le programme affiche la grille
- Le programme demande puis affiche une carte espions pour les que les espions la connaissent
- Une touche permet de cacher la carte espion

Cahier des charges 2 : 2

- On peut choisir une case, la case est alors cachée et remplacée par le numéro de l'équipe à laquelle elle correspond

Cahier des charges 3 : 3

- Gestion automatique des tours de jeu

15 Tirage *sans remise*

Idée générale

On veut tirer des éléments **sans jamais retomber deux fois sur le même** (par exemple, des cartes d'un paquet). Le principe le plus simple et robuste consiste à :

1. Stocker tous les éléments dans un tableau (paquet).
2. À chaque tirage k, **tirer un index aléatoire** parmi les positions encore disponibles [0 .. `borne`].
3. **Échanger** l'élément tiré avec celui en fin de zone disponible (`borne`), puis **réduire la borne** d'1.

Ainsi, la zone `[0..borne]` contient encore les cartes disponibles, et la zone `[borne+1..fin]` contient les cartes déjà tirées.

Schéma visuel pas à pas

Exemple avec 6 cartes initiales : `[A, B, C, D, E, F]` (indices `0..5`), `borne = 5`.

Tirage #1 on tire un index aléatoire `i = 2` (la carte `C`) dans `[0..5]`.

0	1	2	3	4	5
A	B	C	D	E	F

On **échange** la case `i=2` avec la case `borne=5` :

0	1	2	3	4	5
A	B	F	D	E	C

Puis `borne ← 4`. La carte `C` est « sortie » (à droite), elle ne sera plus retirable.

Tirage #2 on tire `i = 1` dans `[0..4]` (car `borne=4`) :

0	1	2	3	4	5
A	B	F	D	E	C

On **échange** avec `borne=4`, puis `borne ← 3` :

0	1	2	3	4 (tirée)	5 (tirée)
A	E	F	D	B	C

On poursuit ainsi jusqu'à ce que `borne < 0` (toutes les cartes tirées), ou qu'on ait tiré le nombre voulu de cartes.

Erreurs fréquentes à éviter

- **Ne pas réduire la borne** ⇒ risque de re-tirer la même carte.
- **Tirer sur toute la longueur** `[0..N-1]` à chaque fois ⇒ doublons possibles.