Première partie

Affichage

Chapitre 1

Menu.c

```
void mainMenu(){
    /*
        Afficher menu principal
        1 - Nouvelle Partie
        2 - Charger une Partie
        3 - Quitter
     */
}

void gameMenu(short noPlayer, movable, attackable){
    /*
        Afficher le menu de jeu
        1 - Unite pouvant se deplacer -> unitMenu();
        2 - Unite pouvant attaquer -> unitMenu();
        3 - Changer de direction -> unitMenu();
        4 - Passer tour
        5 - Abandonner la partie
     */
}

void unitMenu(int choice, short noPlayer, movable, attackable){
    /*
        Afficher la liste des unites pouvant faire quelque chose
        Si choice-> 1 alors liste des unites pouvant se deplacer
        Si choice -> 2 alors liste des unites pouvant attaquer
        Si choice -> 3 alors liste de toutes les unites du joueur
        Appui sur une touche fait appel a playTurn();
     */
}
```

Chapitre 2

Grid.c

```
void gridDisp(){
    // Affiche la grille
}

void dispX(){
    // Affiche les coordonnees horizontales
}

void dispTile(unitName name){
    // Affiche une case
}
```

Deuxième partie

Moteur de jeu

Chapitre 3

gameEngine.c

```
bool gameInit(short * noPlayer){
    gridInit(); // Retourne un code d'erreur
    playerInit(); // Retourne un code d'erreur
    * noPlayer = rand(1,2); // Choisis le joueur qui commence en
        premier
    return true; // Si les fonctions ne retournent rien
}

void playersInit(){
    // Initialise les joueurs (listes + placement unites)
}

void playerAddUnit(short noPlayer, int * nbUnit){
    // Placement des unites par le joueur en cours
}

void gridInit(){
    // Initialise la grille
}

void selectUnit(vector * unitSelected, short noPlayer){
    // Selectionne l'unite -> coordonnees dans la matrice sous
        forme de vecteur
}

void playTurn(short noPlayer){
    unitAction movableUnits[NB_MAX_UNIT]; // Tableau des unites
        pouvant se deplacer + deplacement possible

    movable(movableUnits, noPlayer); // Retourne le tableau mis a
        jour + nombre d'unites pouvant se deplacer
    selectUnit(unitSelected, noPlayer); // Renvoie les coordonnees
        de l'unite selectionnee
    // Choisir l'action desiree
    playMove(unitSelected, movableUnits);
    playAttack(unitSelected);
    playDirection(unitSelected);
}

bool endGame(short noPlayer){
    /*
        Test les conditions de victoire ou de match nul
        Affiche le vainqueur le cas echeant
    */
}
```

```
int movable(unitAction movableUnits[], short noPlayer){
    /*
        Retourne nombre d'unites deplacable + liste maj des unites
            pouvant se deplacer
    */
}

void timeTurn(){
    // Gestion du temps
}

bool lookAround(vector currentUnit){
    /*
        Regarde autour de l'unite selectionnee si elle peut se
            deplacer sur une case si TP pas permise alors
            deplacement impossible
     */
}

void playMove(vector unitSelected, unitAction path[]){
    /*
        Deplace l'unite selectionnee a l'endroit desire en prenant
            en compte les chemins possibles
     */
}

void playAttack(vector unitSelected){
    /*
        Attaque avec l'aide de l'unite selectionnee une unite
            contenu dans la liste listUnitP
     */
}

void playDirection(vector unitSelected){
    //Change la direction de l'unite selectionnee
}
```

Chapitre 4

Grid.c

```
void gridDisp(){
    // Affiche la grille
}
```

Troisième partie

Gestion des chaînes de caractères

Chapitre 5

manageString.c

```
void getCoordS(char coordString[], vector * coordUnit){
    // Recupere les coordonnees de l'unite a partir d'une chaine
}

char* get2Char(char name[]){
    // Recupere 2 caracteres pour l'affichage du nom de l'unite
}

char* getNameUnit(unitName name){
    // Recupere le nom de l'unite a partir de la liste enumeree
}

void printNameUnit(unitName name){
    // Affiche le nom de l'unite
}

void isOutGrid(char * coordString){
    // Verifie que les coordonnees sont dans la grille
}

void correctCoord(char * coordString){
    // Verifie l'authenticite des coordonnees
}

void rightSide(char * coordString, short noPlayer){
    // Verifie que les coordonnees correspondent au bon camp
}

void clearBuffer(){
    // vide le tampon memoire
}

int read(char * string, short length){
    // Lecture securisee d'une chaine de caracteres
}
```

Quatrième partie

Gestion du terminal

Chapitre 6

terminal.c

```
char * getColor(int color, char type[]){
    // Recupere le code correspondant a la couleur
}

void color(int color, char type[]){
    // Change la couleur de l'ecran ou de la police
}

void fontColor(int color){
    // Change la couleur de la police
}

void clearScreen(){
    // Efface l'ecran
}
```

Cinquième partie

Gestion des listes

Chapitre 7

listes.c

```
void addUnit(short noPlayer, vector coordUnit){
    // Ajoute une unite dans la liste du joueur concernee
}

void printList(short noPlayer){
    // Affiche la liste des unites du joueur correspondant
}

void destroyUnit(short noPlayer, vector coordUnit){
    // Detruit une unite dans la liste du joueur correspondant
}
```

Sixième partie

Gestion des unités

Chapitre 8

unit.c

```
void unitInit(short noPlayer, vector coordUnit){
    // Initialise l'unite venant d'etre ajoutee
}

bool canGetPassed(unit * target){
    // Renvoie faux si l'unite ne peut etre traversee
}

bool canBlock(unit * target){
    // Renvoie faux si l'unite ne peut pas bloquer
}

bool canAttack(unit * target){
    // Renvoie faux si l'unite ne peut attaquer
}

bool canMove(unit * target){
    // Renvoie faux si l'unite ne peut bouger
}

void heal(vector source, short noPlayer){
    // Heal toutes les unites du joueur courant
}

int getSideAttacked(vector source, vector target){
    // Renvoie le cote cible par l'unite source
}

void attack(vector source, vector target){
    // Attaque l'unite cible avec l'unite source
    // Prendre en compte les capacites speciales
}

void copy(unit * destination, unit * source){
    // Copie l'integralite d'une structure vers une autre
}

void move(vector destination, vector source){
    // Deplace l'unite source vers le vecteur destination
}

void setDirection(vector source, short dir){
    // Definis la direction de l'unite
}
```

```
void addEffect(vector target, unitEffect effect){
    // Ajoute un effet sur l'unite
}

void AoE(vector target, int size, int dmg, bool own){
    // Attaque de zone
}

void line(vector source, int size, int dmg, int dir){
    // Attaque sur une ligne
}
```