BONNET Émile
MARCHAIS Mattieu

CO5SFIN0 - Introduction à la programmation S5

Compte-Rendu - Projet jeu C#

“JURENSIC WORLD”

Pr. TETELIN Angélique

2024-2025

1. Introduction

a. Demande initiale

Suite aux cours du semestre 5 sur le langage de programmation C#, et des quelques TP d'introduction, il nous a été demandé de réaliser un jeu vidéo sur ce langage pour la fin de l'année. L'objectif est de créer un jeu de plateau en 2D avec des personnages se déplaçant dessus. Nous allons présenter les attendus exacts de ce projet.

Règles du jeu:

"Vous incarnerez 2 personnages dans ce jeu : Owen Grady et la raptor Blue. Ils luttent ensemble pour sauver la petite Maisie qui est poursuivie par l'Indominus Rex."

-Pour gagner une partie vous devrez enfermer L'indominus Rex à l'aide de crevasse. Ces crevasses sont provoquées par les grenades que Owen est capable de lancer. Chaque grenade provoque deux crevasses à côté et Owen possède un nombre limité de grenade. Il ne peut lancer les grenades qu'à trois cases de lui maximum.

-Owen et Blue ne se déplacent que d'une case à la fois. A chaque tour, Owen a le choix entre se déplacer ou lancer une grenade. Blue quant à elle peut faire reculer l'Indominus Rex lorsqu'elle entre en contact avec elle. L'indominus Rex recule alors de 3 cases (ou moins si elle rencontre une crevasse ou un bord du jeu) et est immobilisée pour ce tour.

-Maisie elle n'est pas un personnage jouable, elle panique face à la férocité de l'Indominus Rex et bouge donc aléatoirement d'une case par tour. (Cependant elle ne peut pas se jeter sur l'indominus Rex)

-L'Indominus Rex est attirée par tout le monde, elle ne sait plus où donner de la tête. Ses mouvements sont donc aléatoires. L'indominus peut manger Owen ou Maisie la partie sera alors perdue. L'Indominus Rex ne mange pas ses compères, Blue ne peut donc pas être mangée. De plus l'Indominus a la peau trop dure pour craindre les grenades.

-Ni n'importe quel personnage de votre équipe se prend une grenade, c'est la défaite. Attention donc à bien viser car une grenade provoque 2 crevasses, la première là où Owen l'a lancée et la deuxième aléatoirement entre les 4 positions possibles autour de la première crevasse

-Aucun personnage ne peut bouger en diagonale, donc ils bougeront donc selon les directions haut/bas et gauche/droite

Résumé des règles:

Owen: → Peut bouger d'une case par tour ou alors peut décider de lancer une grenade.

Grenade: → Il y a un nombre limité de grenades.

→ Seul Owen peut lancer des grenades (avec une portée de 3 cases).

→ Une grenade crée 2 crevasses (une crevasse n'est pas franchissable).

Blue: → Peut bouger d'une case par tour

→ Fait reculer l'Indominus Rex de 3 cases et l'immobilise pour le tour (ou moins si elle rencontre une crevasse ou un bord du jeu).

Maisie: → Bouge aléatoirement (mais jamais vers l'Indominus)

Indominus Rex: → Bouge aléatoirement

→ Peut manger Owen et Maisie

→ Ne craint pas les grenades

Condition de victoire: → L'Indominus Rex est enfermée

Condition de pertes: → L'indominus a mangé Owen ou Maisie

→ Owen n'a plus de grenade

→ Owen, Blue ou Maisie se prennent une grenade

Fonctionnalités attendues:

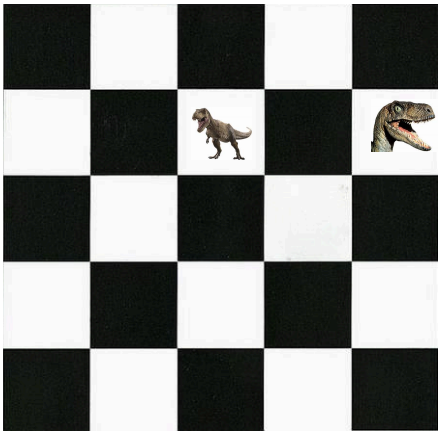
- Construire le plateau de jeu
- Gérer le déplacement des personnages joueurs et des personnages non joueurs à chaque tour : - Owen et Blue doivent pouvoir choisir une direction
  - pour l'Indominus et Maisie les directions doivent être tirées aléatoirement,
  - Aucun personnage ne peut franchir une case qui a « sauté »
- Offrir à Owen la possibilité d'utiliser une grenade à chaque tour et de choisir les coordonnées de la case où elle atterrit dans ses limites de lancement
- Réafficher le plateau après les modifications
- Gérer si c'est gagné/perdu

Note sur les fonctionnalités attendues:

En commençant à réfléchir à la manière d'aborder le jeu et à la façon de coder nous nous sommes rendu compte que certains cas particuliers du jeu n'étaient pas explicites dans les consignes, il fallait donc faire un choix. Nous ne considérons pas ces choix comme des fonctionnalités créatives. C'est pourquoi elles seront détaillées dans la structure du jeu. Toutefois nous aimerions détailler une particularité qui peut poser problème.

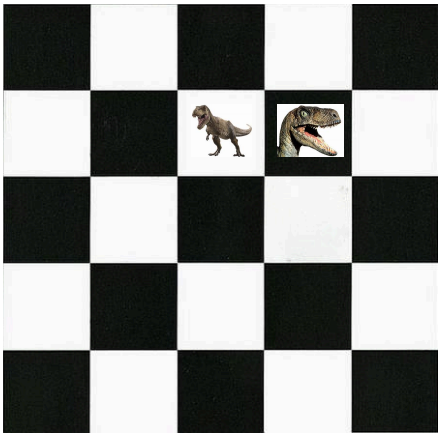
Dans certaines configurations de jeu, il est impossible pour deux personnages d'être sur la même case. Il est donc impossible de provoquer certaines conditions de gameplay (l'Indominus Rex qui mange un personnage, Blue qui fait reculer l'Indominus Rex).

Appuyons nous sur le schéma suivant:

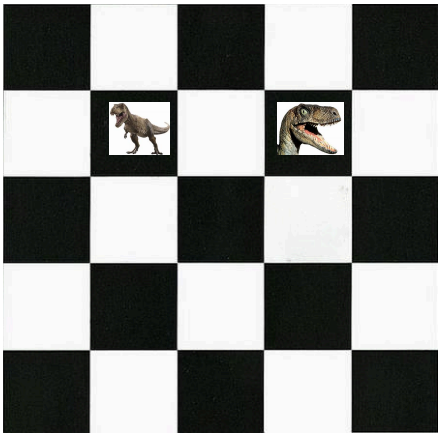


Blue :  Indominus Rex : 

Nous avons considéré (un peu comme pour Maisie avec l'Indominus Rex) que l'Indominus Rex ne se déplaçait jamais sur la même case que Blue. Si l'objectif du joueur est de faire reculer l'Indominus Rex, le joueur devra déplacer Blue vers elle.



Or dans cette situation l'Indominus ne bougera pas vers Blue, elle ira donc sur une case noire et on se retrouvera dans la même situation que précédemment.



A la fin de chaque tour (quand Blue et l'Indominus Rex viennent de bouger), elles se retrouvent toujours sur des cases de même couleur. Elles ne pourront donc jamais se toucher.

Pour résoudre ce problème qui est identique avec n'importe quel personnage devant entrer en collision avec un autre, nous tenterons certaines stratégies que nous expliquerons dans la partie créative.

b. Rappel des contraintes

-Ce projet devra être programmé en C#, en mode console, sous VS Code, dans les conditions habituelles des TP.
-L'utilisation de bibliothèques de fonctions externes autres que les bibliothèques standards du framework .NET ne sont pas acceptées.
-Nous devons programmer avec une approche procédurale, en n'utilisant pas de classes autres que celles vues en CM et TD-TP.
-La version 8 de .NET est la seule acceptée. L'utilisation des listes et autres collections est exclue, le traitement sera basé sur des manipulations de tableaux.

c. Organisation du travail et répartition des tâches

Pour l'organisation du travail, nous avons choisi de commencer en créant le strict minimum demandé pour que nous puissions être plus libre dans le codage de notre partie créative. Pour faire cela, nous avons divisé les tâches à faire entre nous, une par une, puis chaque nouvelle réunion, nous nous mettions d'accord sur les prochaines fonctions à créer. Par exemple, après la première réunion, nous avions décidé que Mattieu travaillerai sur les conditions de défaite et l'initialisation du jeu, et Emile sur le déplacement des personnages et le lancer de grenade. De plus, dès que nécessaire, nous organisions une séance de débogage entre nous si trop d'erreur différentes étaient présente. Avoir une fondation de code fonctionnel aidait grandement ces séances de débogage, car il était donc plus simple d'identifier la source de chaque bug individuellement.

Pour la réalisation de ce rapport et pour diviser notre code en parties ordonnées, nous avons délimité le travail en plusieurs catégories que nous allons détailler (cela concerne les parties 3 à 8).

2. Partie Créative

Ce que nous avons choisi de rajouter en plus de ce qui a été demandé fut:

- Un menu de sélection au début qui permettait d'intégrer les différentes parties créative tout en donnant la possibilité au joueur de jouer au jeu originale, car nous souhaitions que toute fonctionnalité créative qui n'était pas esthétique devait rester optionnel.
- Nous avons choisi de rajouter différents modes de difficultés, notamment pour palier au problème expliquer précédemment concernant le fait que certain personnages ne pourrons jamais se toucher. Ainsi, dans le mode de jeu Difficile, une fois sur 5, Indominus Rex bouge 2 fois. De plus son déplacement n'est plus aléatoire mais est attiré par les humains sur le plateau, se dirigeant constamment vers soit Owen soit Maisie, en fonction du quel est le plus proche. Dans le mode de jeu Impossible, Indominus Rex se déplace deux fois à chaque tour, en restant attiré par les humains.
- Il est possible de rendre le déplacement de Blue intelligent sous quel cas vous ne la controlez plus.

- Une arme additionel est aussi disponible pour les difficultés Difficile et Impossible: un filet. Il ne peut être lancé que quand Indominus Rex est à deux case ou moins de Owen.
- Chaque personnage est affiché d'une couleur différente pour mieux les identifiés sur le plateau. Les crevasses sont aussi affichés en couleur.
- Il est aussi possible de jouer le jeu "inversé", où vous jouez l'Indominus Rex qui essaie de manger Owen et Maisie. Dans ce mode, chaque tour, une case aléatoire devient une crevasse, Blue se déplace toujours vers l'Indominus Rex et Owen et Maisie fuit constamment l'Indominus Rex.

Toutes les composantes créatives de notre code dans la suite de notre rapport sont surligné en rose ainsi.

3. Initialisation de toutes les variables et objets

a. Paramètres globales

```
int lengthGridX = 10;
int lengthGridY = 10;
```

Ces deux paramètres définissent la taille du plateau. Elles sont définient dans le main car elles doivent être appelé dans plusieurs fonctions différentes.

```
string nextPrint = "";//Servira pour le placement du curseur au milieu lors de l'affichage du texte
```

Nous permet quand nous affichons du texte de l'afficher au milieu. Il faut que nous l'appelions dans chaque fonction qui imprime du texte donc se doit d'être dans le main.

```
bool playableBlue = true;// Booléen qui permet de contrôler Blue ou non
```

Variable qui nous permet de contrôler si Blue est controlé par l'ordinateur ou par le joueur. Elle est dans le main car plusieurs fonctions sont dépendante de cette variable.

```
string difficulty = "Normal";
```

Variable du choix de la difficulté. Elle est dans le main car comme pour la variable du contrôle de Blue, plusieurs fonction l'utilise comme variable.

```
int selectNumber = 1;
```

Variable qui determine sur quelle option le joueur est à chaque moment. Cela nous permet d'avoir un affichage qui ne demande pas à l'utilisateur de rentrer quique ce soit dans la console.

```
int chooseOption = 1;
```

Cette variable à la même utilité que selectNumber mais pour la page des options.

```
char[,] trenches = new char[lengthGridY, lengthGridX];
for (int i = 0; i < lengthGridY; i++)
{
    for (int j = 0; j < lengthGridX; j++)
    {
        trenches[i, j] = '.';
    }
}
```

Permet d'initialiser la grille en la remplissant de notre caractère neutre, un point. Cette variable contient la position des crevasses, cependant puisque elle est initialisé avant, elle est utilisée pour pour définir les deux autres grilles.
Se doit d'être dans le main car elle est appelés pour toutes les fonctions qui utilisent les crevasses.

```
char[,] colorBackground = new char[lengthGridY, lengthGridX];
```

Cette variable est la grille qui contient la position des crevasses et qui est tuilisé pour la condition de victoire. Elle est dans le main car il est nécéssaire que elle soit globale pour permettre a plusieurs fonction de l'appeler et en dépendre.

```
char[,] grid = new char[lengthGridY, lengthGridX];
```

Cette variable est la variable qui contient la grille de jeu. Elle est dans le main pour les mêmes raison que les deux autres grilles. Elle est dans le main pour la même raison que colorBackground.

```
/* Initialisation Owen*/
char owen = 'O';
int owenPositionX = lengthGridX - rnd.Next(1, 3);
int owenPositionY = rnd.Next(1, 3);
int nbGrenade = (lengthGridX + lengthGridY) / 2;
int nbNet = (lengthGridX + lengthGridY) / 5;

/* Initialisation de l'objet grenade*/
char grenade = 'G';
int grenadePositionX = -1;
int grenadePositionY = -1;

/* Initialisation Blue*/
char blue = 'B';
int bluePositionX = rnd.Next(1, 3);
int bluePositionY = lengthGridY - rnd.Next(1, 3);

/* Initialisation Indominus Rex*/
char indominusRex = 'I';
int indominusRexPositionX = lengthGridX - rnd.Next(1, 3);
int indominusRexPositionY = lengthGridY - rnd.Next(1, 3);
int indominusRexNoMove = 0;

/* Initialisation Maisie*/
char maisie = 'M';
int maisiePositionX = rnd.Next(1, 3);
int maisiePositionY = rnd.Next(1, 3);
```

Initialisation de toutes les variables qui concernent chaque personnage individuellement. Owen est généré dans le coin

Nord-Est, Blue dans le coin Sud-Ouest, Maisie dans le coin Nord-Ouest et l'Indominus Rex dans le coin Sud-Est.
Nous générons la position de chaque personnage aléatoirement dans un carré de 2x2 par rapport à leur coin respectif.

b. Fonctions pour l'interface de départ

```
> void PrintIntro()//-> Affiche l'introduction du jeu
```

Paramètres d'entrée:

Description: La fonction imprime l'ASCII du T-Rex, puis affiche le texte d'introduction. Il demande de rentrer un char pour pouvoir passer à l'écran suivant.

Paramètres de sortie:

```
void PrintSelectScreen(int x)//-> Affiche l'écran de selection en fonction de l'entier x (il y a 4 possibilités)
```

Paramètres d'entrée: (un int correspondant à la séléction du menu)

Description: La fonction à tout le texte à afficher pour le menu principal. Elle est diviser avec des if en fonction de la séléction du joueur.

Paramètres de sortie:

```
void SelectScreen()//-> Permet la selection des différents écrans de selection
```

Paramètres d'entrée:

Description: La fonction fait bouger la selection de menu du joueur en fonction de 3 touches, z et s pour la navigation et espace pour confirmer.

Paramètres de sortie:

```
void ChooseLengthGrid()//-> Permet de choisir la taille de la grille
```

Paramètres d'entrée:

Description: La fonction demande la taille de grille au joueur et la definie de manière globale en appelant lengthGridX/Y. La fonction comporte aussi une sécurité car elle n'accepte que les entiers entre 5 et 40. Elle n'accepte pas non plus autre chose que des int.

Paramètres de sortie:

```
void PrintSelectOptions(int chooseOption)//-> Affiche l'écran des options en fonction de l'entier x (il y a 3 possibilités)
```

Paramètres d'entrée: (un int correspondant à la séléction des options)

Description: La fonction fonctionne exactement comme PrintSelectScreen, seulement qu'elle imprime les différentes options.

Paramètres de sortie:

```
void Options()//-> Permet la selection des différents écran de selection
```

Paramètres d'entrée:

Description: La fonction fonctionne comme SelectScreen, mais permet de changer les variables d'options concernées.

Paramètres de sortie:

```
void PrintColoredText(string input)//-> Affiche un texte avec les couleurs pour chaque personnage
```

Paramètres d'entrée: (un string)

Description: La fonction sert à remplacer le WriteLine dans le cas ou le texte imprimé change de couleur régulièrement. Il prend un texte en valeur, le sépare grâce à la fonction Split, et utilise un foreach pour changer de couleur en fonction de chaque mot que l'on veut d'une couleur spécifique.

Paramètres de sortie:

```
void Rules()//-> Affiche les règles du jeu
```

Paramètres d'entrée:

Description: La fonction contient tout les strings que nous voulons imprimer, et ultilise PrintColoredText pour l'imprimer de couleur dans la page Règle de l'écran de séléction. Il faut rentrer un char quelquonque pour revenir au menu principal.

Paramètres de sortie:

```
void InterfaceJeu()//-> Effectue toutes les interfaces et actions avant le lancement réel du jeu (Ecran de sélection/Options/Règles)
```

Paramètres d'entrée:

Description: La fonction rassemble toutes les fonctions décrites précedemments pour lancer une première partie du jeu. Un while loop permet de rester dans l'écran de séléction tant que le joueur n'as pas choisi de jouer.

Paramètres de sortie:

---

c. Fonctions pour la grille

```
void DefineColorBackground()    //-> créer une grille identique aux crevasses pour l'utiliser dans la condition de victoire
```

Paramètres d'entrée:

Description: La fonction fonctionne exactement comme le if loop montrer plus tôt où la liste est rempli de caractères neutres qui pour nous sont '.' .

Paramètres de sortie:

```
void DefineGrid()   //-> Permet de mettre à jours la grille lors des déplacements des différents personnages
```

Paramètres d'entrée:

Description: La fonction fonctionne exactement comme le if loop montrer plus tôt où la liste est rempli de caractères neutres qui pour nous sont '.' .

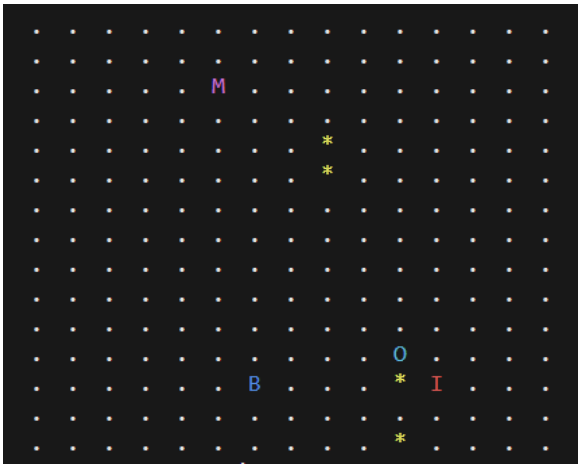
Paramètres de sortie:

---

4. Affichage de la grille

a. Représentation de la grille

Pour l'affichage de la grille nous avons tout d'abord considéré le plateau de jeu avec une matrice de caractères en deux dimensions. Une case vide correspond à un point, chaque personnage est représenté par la première lettre de son nom. et une crevasse est représenté par une étoile " * ".



b. Définition des personnages et objets

Avant l'affichage de la grille pour remettre à jour les positions de chaque personnage. Pour cela, nous avons créer les fonctions :
DefineOwen, DefineBlue, DefineMaisie, DefineGrenade et DefineIndominusRex.

```
void DefineOwen(int x, int y) //-> Permet de mettre à jour la position d'Owen sur la grille lors de son affichage
{
    if ((x == owenPositionX) && (y == owenPositionY))
    {
        grid[y, x] = owen;
    }
}
```

Paramètres d'entrée: (deux entiers correspondant à des positions)

Description: La fonction test si les positions entrées en paramètre sont égales aux positions de Owen. Ainsi on met à jour la position de Owen dans la grille

Paramètres de sortie:

Les 4 autres fonctions sont fondées sur le même principe. Nous ne les détaillerons donc pas.

```
void DefineMaisie(int x, int y) //-> Permet de mettre à jour la position de Maisie sur la grille lors de son affichage
void DefineIndominusRex(int x, int y) //-> Permet de mettre à jour la position de l'IndominusRex sur la grille lors de son affichage
void DefineGrenadeG(int x, int y) //-> Permet d'afficher l'endroit où on lance la grenade sur la grille lors de son affichage
void DefineBlue(int x, int y) //-> Permet de mettre à jour la position de Blue sur la grille lors de son affichage
```

```
void DefineCaracters(int x, int y)//-> Cette procédure réunit les 5 dernières en une seule
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description: Cette fonction permet simplement de faire appel aux 5 fonctions précédentes en même temps.

Paramètres de sortie:

---

c. Fonction d'affichage

```
void ShowGrid() //-> Affichage de la grille
{
```

Paramètres d'entrée:

Description:

-D'abord on redéfinit les positions des crevasses avec *DefineGrid.*
-On vide la console.
-On parcourt ensuite toute la grille pour l'afficher. Cependant pour chaque case de la grille, on fait appel aux 5 fonctions *Define{…}* pour mettre à jour les positions.
-On teste ensuite quel caractère il faut afficher pour changer la couleur du curseur.
-Enfin on affiche le caractère en question.

Paramètres de sortie:

5. Déplacements et actions des joueurs, objets et PNJ

Dans cette partie, nous expliquerons les fonctions concernant le mouvement des personnages et aussi les actions de ses derniers

a. Owen

```
void MoveOwen() //-> Pour faire bouger Owen et lancer ses grenades
```

Paramètres d'entrée:

Description:
-On affiche d'abord les instructions de déplacements et d'actions possibles.
-Pour un déplacement, on prend la valeur rentrée par l'utilisateur et test si le déplacement est valide.
-Pour une action, on fait appel à la fonction correspondante:
ThrowGrenade()
ThrowNet()
-On se déplace ou provoque une action en conséquence.
-On met à jour les positions et actualise la grille.

Paramètres de sortie:

```
bool IsMoveOwenBlueValid(int x, int y)//-> Test si le mouvement est possible pour Owen et Blue
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description:
-On teste si les positions en paramètre sont valide pour Owen et Blue. Ils ne peuvent pas se déplacer en dehors de la grille, vers des crevasses ou vers Maisie.

Paramètres de sortie: Renvoie un booléen (True si déplacement valide/ False sinon)

```
void MoveOwenPNJ() //-> Pour déplacer Owen en temps que non joueur
```

Paramètres d'entrée:

Description:

-On sort une direction de déplacement qui est d'abord intelligente (Owen fuit l'Indominus Rex). Si cela n'est pas possible, il sort une direction aléatoire jusqu'à ce que se soit possible.
-On se déplace en conséquence
-On met à jour les positions

Paramètres de sortie:

```
void ThrowGrenade() //-> Pour lancer une grenade d'Owen
```

Paramètres d'entrée:

Description:
-Lorsque la fonction est appelée, un caractère G apparaît sur Owen que l'on peut déplacer comme pour tous les autres personnages jouables.
-On affiche les instructions de déplacements, l'utilisateur peut décider de revenir à son déplacement s'il change d'avis ou s'est trompé.
-Si le mouvement est valide (c'est à dire pas en dehors de la grille et qui reste dans un carré de 3 cases autour de Owen), on déplace le caractère en conséquence
-Lorsque que l'utilisateur confirme son lancer, il fait appel à la fonction *PlaceGrenade()* qui va placer les crevasses à la position du caractère "G"

Paramètres de sortie:

PS: On peut considérer les deux fonctions précédentes comme "créatives" dans le sens ou le choix de l'emplacement du lancer se fait par un déplacement d'un caractère plutôt que par un choix de coordonnées comme demandé initialement. Nous avons fait ce choix car cela fluidifie le déroulement du jeu.

```
void PlaceGrenade() //-> Place la grenade sur la grille au moment de la confirmation du lancer
```

Paramètres d'entrée:

Description:
-Le nombre de grenade diminue de 1 et on place une crevasse dans la grille pour les crevasses à la position du caractère "G"
-On sort un chiffre aléatoire entre 1 et 4 pour déterminer le placement de la deuxième crevasse
-On fait appel à la fonction *DefineColorBackground* qui sert pour la condition de victoire. Nous y reviendrons.
-On fait disparaître le caractère "G" de la grille

Paramètres de sortie:

```
void ThrowNet() //-> Pour lancer un filet
```

Paramètres d'entrée:

Description:

-On affiche les instructions du lancer du filet, l'utilisateur peut décider de revenir à son déplacement s'il change d'avis ou s'est trompé.

-S'il confirme le nombre de filet à disposition diminue et on immobilise l'Indominus Rex pendant 2 tours

Paramètres de sortie:

b. Blue

```
void MoveBlue() //-> Pour faire bouger Blue et reculer l'IndominusRex
```

Paramètres d'entrée:

Description:

-On affiche d'abord les instructions de déplacements et d'actions possibles.

-Pour un déplacement, on prend la valeur rentrée par l'utilisateur et test si le déplacement est valide.

-On se déplace en conséquence

-Pour l'action de Blue, on fait directement appel en fin de déplacement à la fonction *StepBackIndominusRex()* qui fait reculer l'Indominus Rex. C'est cette fonction qui testera si l'action est possible ou pas

-On met à jour les positions et actualise la grille.

Paramètres de sortie:

```
void MoveBluePNJ() //-> Pour déplacer Blue quand on ne la contrôle pas
```

Paramètres d'entrée:

Description:

-On sort une direction de déplacement qui correspond à un chiffre et test si ce déplacement est valide (Blue se déplace toujours vers l'Indominus Rex)

-On se déplace en conséquence

-On fait appel à la fonction *StepBackIndominusRex()*

-On met à jour les positions

Paramètres de sortie:

```
void StepBackIndominusRex(char action)//-> Fait reculer l'indominusRex quand Blue la touche
```

Paramètres d'entrée: Un caractère qui correspond au dernier mouvement de Blue. Il sert à savoir dans quelle direction faire reculer l'Indominus Rex

Description:
-On fait reculer l'Indominus en regardant d'où provient Blue
-On fait attention à ce que l'Indominus Rex ne sorte pas de la grille ou ne traverse pas une crevasse

Paramètres de sortie:

c. Maisie

```
void MoveMaisie() //-> Pour déplacer Maisie
```

Paramètres d'entrée:

Description:
-On sort une direction de déplacement qui correspond à un chiffre et test si ce déplacement est valide
-On se déplace en conséquence
-On met à jour les positions

Paramètres de sortie:

```
bool IsMoveMaisieValid(int x, int y)//-> Test si le mouvement est possible pour Maisie
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description:
-On teste si les positions en paramètre sont valides pour Maisie. Elle ne peut se déplacer que vers des cases vides.

Paramètres de sortie: Renvoie un booléen (True si déplacement valide/ False sinon)

```
void MoveMaisieSmart() //-> Pour déplacer Maisie de façon intelligente
```

Paramètres d'entrée:

Description:
-On sort une direction de déplacement qui est d'abord intelligente (Maisie fuit l'Indominus Rex). Si cela n'est pas possible, on sort une direction aléatoire jusqu'à ce que ce soit possible.
-On se déplace en conséquence
-On met à jour les positions

Paramètres de sortie:

d. Indominus Rex

```
void MoveIndominusRex() //-> Pour déplacer IndominusRex
```

Paramètres d'entrée:

Description:

-On regarde le mode de difficulté choisi pour savoir si elle se déplace intelligemment ou pas

-On sort une direction de déplacement qui correspond à un chiffre et test si ce déplacement est valide

-On se déplace en conséquence

-On met à jour les positions

Paramètres de sortie:

```
bool IsMoveIRValid(int x, int y)//-> Test si le mouvement est possible pour l'Indominus Rex
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description:

-On teste si les positions en paramètre sont valides pour l'Indominus Rex. Elle ne peut pas se déplacer en dehors de la grille, vers des crevasses ou vers Blue.

Paramètres de sortie: Renvoie un booléen (True si déplacement valide/ False sinon)

```
void MoveIndominusRexPlayer() //-> Pour faire bouger L'Indominus Rex
```

Paramètres d'entrée:

Description:

-On affiche d'abord les instructions de déplacements et d'actions possibles.

-Pour un déplacement, on prend la valeur rentrée par l'utilisateur et test si le déplacement est valide.

-On se déplace en conséquence

-On met à jour les positions et actualise la grille.

Paramètres de sortie:

6. Conditions de victoire ou de pertes

Cette partie du code n'initialise qu'une seul variable globale:

```
int conditionWinLose = 0; //Variable qui retient quelle condition d'arrêt a été validée
```

qui est la condition qui vérifie quelle condition de victoire/défaite à été validé, pour pouvoir afficher le bon texte en réponse.

a. Condition de victoire

```
void IndominusRexPossibilities(int x, int y)//-> Cherche toutes les cases atteignables par l'IndominusRex
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description: La fonction est une fonction récursive de remplissage en partant de la position de l'Indominus Rex pour vérifier toutes les cases qu'elle peut atteindre. Elle se rappelle elle-même constamment pour toute les cases voisines à condition qu'elles soient accessible.

Paramètres de sortie:

```
bool CheckPosition()//-> Vérifie que les cases atteignables par l'Indominus Rex ne contiennent pas les autres personnages
```

Paramètres d'entrée:

Description:

- On teste si parmis les cases atteignable par l'Indominus Rex, il y as un des autres personnages.

Paramètres de sortie: Renvoie un booléen (True si elle est seul/ False si un autre PNJ est atteignable)

```
bool CheckWin()//-> Regarde si la condition de victoire est vérifiée (Indominus Rex enfermée)
```

Paramètres d'entrée:

Description: Cette fonction regroupe les deux autre fonctions pour vérifier si elle est enfermée dans une seul fonction.

Paramètres de sortie: Renvoie un booléen (False si elle est enfermé/ True sinon)

---

b. Conditions de défaite

```
bool LosingConditionGrenadePerdu()//-> Regarde les conditions de perte concernant la grenade lancée sur d'autre
```

Paramètres d'entrée:

Description: Vérifie que le joueur n'as pas envoyé une grenade sur un des personnages gentils en comparant la position de chacun avec la position des grenades.

Paramètres de sortie: Renvoie un booléen (False si un gentil à reçu une grenade/ True sinon)

```
bool LosingConditionGrenade()//-> Regarde les conditions de perte concernant le nombre de grenade
```

Paramètres d'entrée:

Description: Vérifie qu'il reste des grenades au joueur.

Paramètres de sortie: Renvoie un booléen (False si il n'en as plus/ True sinon)

```
bool LosingConditionManger()//-> Regarde les conditions de perte concernant l'IndominusRex qui mange les joueurs
```

Paramètres d'entrée:

Description: Similairement à la vérification pour les grenades, il vérifie que la position de l'Indominus Rex n'est pas aussi celle de Owen ou Maisie.

Paramètres de sortie: Renvoie un booléen (False si l'Indominus a mangé un humain/ True sinon)

---

c. Affichage textuelle.

```
void TexteWinLose()//-> Affiche les texte une fois que l'on a gagné/perdu en fonction de la cause.
```

Paramètres d'entrée:

Description: La fonction vérifie quelle condition à arreté le jeu et affiche le texte qui va avec. Il affiche également la dernière version de la grille, tout en enlevant le PNJ ou le joueur qui as été touché par une grenade ou mangé.

Paramètres de sortie:

```
void TexteWinLoseAlternatif()//-> Affiche les texte une fois que l'on a gagné/perdu en fonction de la cause.
```

Paramètres d'entrée:

Description: La fonction est la même que pour le jeu de base sauf que le texte en lui-même change, car elle concerne le jeu inversé.

Paramètres de sortie:

---

7. Programmes en tout genre

```
void PrintAscii()//-> Affiche un motif de dinosaure
```

Paramètres d'entrée:

Description: La fonction sert simplement à afficher ligne par ligne l'ASCII du T-Rex.

Paramètres de sortie:

```
char ClosestPlayer(int x, int y)//-> Recherche l'humain le plus proche
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description: La fonction calcule quel humain entre Owen et Maisie est le plus proche.

Paramètres de sortie: un char qui représente quel humain est le plus proche

---

```
int SmartIR(char a)//-> Choisie le déplacement vers l'humain le plus proche
```

Paramètres d'entrée: un char qui représente quel humain est le plus proche

Description: La fonction choisie comment se diriger vers l'humain le proche en calculant si il faut se déplacer verticalement ou horizontalement, en fonction du quel est le plus long.

Paramètres de sortie: Un int qui représente quel direction final est choisie

---

```
int SmartBlue()//-> Choisie le déplacement vers l'Indominus Rex
```

Paramètres d'entrée:

Description: La fonction determine comme pour SmartIR comment se diriger vers IR le plus rapidement possible en fonction de la distance horizontal ou vertical.

Paramètres de sortie: Un int qui représente quel direction final est choisie

---

```
int SmartOwenMaisie(int positionX, int positionY)
```

Paramètres d'entrée: Deux entiers x et y représentant des positions

Description: La fonction est la même que pour Smart Blue seulement la direction est inversé pour constamment fuir l'IR. Elle prend des coordonnées en fonction car cette fonction est utilisé et pour Owen et pour Maisie.

Paramètres de sortie: Un int qui représente quel direction final est choisie

---

```
void RandomGrenades()
```

Paramètres d'entrée:

Description: La fonction choisi aléatoirement une case libre qui devient une crevasse. Elle est reservé au jeu alternatif où vous jouez l'Indominus Rex.

Paramètres de sortie:

---

8. Lancement du jeu

a. Fonctions pour lancer le jeu

```
void MainGame() //-> Pour lancer le jeu en effectuant les différentes actions dans l'ordre
```

Paramètres d'entrée:

Description: La fonction fait tourner le jeu. Elle est composé d'un while loop qui a 4 condition pour tourner, les 3 conditions de défaite et la condition de victoire. Il y as 4 if dans le jeu en soit, qui sont tous responsable de faire tourner les fonctionnalités optionnelles. Le mouvement des PNJ et du joueur est structuré tel que Owen bouge, puis Blue, puis Maisie, puis l'Indominus Rex. Les conditons sont actualisé en dessous de chaque mouvement qui serait responsable de la condition de en question. En dehors du while loop, nous affichons le texte de défaite ou victoire pertinent.

Paramètres de sortie:

```
void MainGameAlternatif() //-> Pour lancer le jeu en effectuant les différentes actions dans l'ordre
```

Paramètres d'entrée:

Description: La fonction est très similaire à MainGame sauf qu'elle est plus simple car elle ne demande aucun if à l'intérieur. Elle change également l'ordre de mouvement qui devient : Indominus Rex, Owen, Blue, Maisie.

Paramètres de sortie:

---

b. Code présent dans le main

```
InterfaceJeu(); // Lancement d'interface avant la définition du plateau de jeu pour les variables lengthGridX/Y
```

Cette fonction est techniquement présente dans notre région Initialisation, mais il était plus pertinent d'en parler ici. Elle permet d'initialiser toute les variables globales correctement avant de jouer, et se doit donc d'être lancer avant les variables en question.
Cette fonction rempli celle que nous avons décrite pour elle plus tôt.

```
if (selectNumber == 3)
{
    MainGameAlternatif();
}
else
{
    MainGame();
}
```

Voici le code qui fait tourner notre jeu en fonction de quel jeu le joueur à choisie. Si toutes nos fonctions serait stocker dans un autre fichier et était appeler dans celui-ci, notre code consisterai de ces lignes ci:

```
Random rnd = new Random();
Console.Clear(); //Réinitialisation de la console
```

```
int lengthGridX = 10;
int lengthGridY = 10;
```

```
string nextPrint = "";//Servira pour le placement du curseur au milieu lors de l'affichage du texte
```

```
bool playableBlue = true;// Booléen qui permet de contrôler Blue ou non
```

```
string difficulty = "Normal";
```

```
int selectNumber = 1;
```

```
int chooseOption = 1;
```

```
char[,] trenches = new char[lengthGridY, lengthGridX];
for (int i = 0; i < lengthGridY; i++)
{
    for (int j = 0; j < lengthGridX; j++)
    {
        trenches[i, j] = '.';
    }
}
```

```
char[,] colorBackground = new char[lengthGridY, lengthGridX];
```

```
char[,] grid = new char[lengthGridY, lengthGridX];
```

```
InterfaceJeu(); // Lancement d'interface avant la définition du plateau de jeu pour les variables lengthGridX/Y
```

```
/* Initialisation Owen*/
char owen = 'O';
int owenPositionX = lengthGridX - rnd.Next(1, 3);
int owenPositionY = rnd.Next(1, 3);
int nbGrenade = (lengthGridX + lengthGridY) / 2;
int nbNet = (lengthGridX + lengthGridY) / 5;

/* Initialisation de l'objet grenade*/
char grenade = 'G';
int grenadePositionX = -1;
int grenadePositionY = -1;

/* Initialisation Blue*/
char blue = 'B';
int bluePositionX = rnd.Next(1, 3);
int bluePositionY = lengthGridY - rnd.Next(1, 3);

/* Initialisation Indominus Rex*/
char indominusRex = 'I';
int indominusRexPositionX = lengthGridX - rnd.Next(1, 3);
int indominusRexPositionY = lengthGridY - rnd.Next(1, 3);
int indominusRexNoMove = 0;

/* Initialisation Maisie*/
char maisie = 'M';
int maisiePositionX = rnd.Next(1, 3);
int maisiePositionY = rnd.Next(1, 3);
```

```
int conditionWinLose = 0; //Variable qui retient quelle condition d'arrêt a été validée
```

```
if (selectNumber == 3)
{
    MainGameAlternatif();
}
else
{
    MainGame();
}
```

9. Bilan du projet

Pour conclure sur ce projet. Bien que chacun de nous avions de solides bases en programmation, coder un jeu était une première pour notre binôme. Le travail était stimulant et nous sommes restés motivés tout au long du projet.

Globalement la structure du jeu, les fonctions que l'on avait à créer ne nous ont pas posé de sérieux problèmes. Petite exception pour l'algorithme de test qui permet de si l'Indominus Rex est enfermée. Cela a été compliqué à mettre en place. Nous avons dû sérieusement réfléchir à un algorithme fonctionnel. Et même une fois le concept trouvé, la programmation de celui-ci nous a donné du fil à retordre (notamment la fonction *IndominusRexPossibilities*).

Nous avions pour objectif de rapidement faire la partie "obligatoire" du jeu pour se concentrer ensuite sur la partie créative. Cependant nous avons mis plus de temps que prévu surtout pour déboguer le programme. Nous avons tout de même eu le temps d'ajouter de nombreux éléments dans cette partie créative.

Ce que nous avons le plus apprécié dans le projet était justement la possibilité de créativité qui nous stimulait en continue (parfois trop, nous avons dû calmer nos ardeurs pour respecter la date limite du projet). Nous aurions aimé par exemple ajouter un joueur supplémentaire, faire une grille plus grande et plus jolie pour faire de vrais personnages au lieu de simples caractères... Nous sommes cependant très contents de la partie créative que nous avons rendu.

Ce qui a été le plus difficile et ce que nous avons le moins apprécié fut de comprendre l'origine des bugs. L'étape de chercher où est l'erreur dans le programme est souvent la plus longue et frustrante. Ce n'est cependant qu'un léger point négatif au vu du plaisir que nous avons eu à réaliser ce projet.