BORDEAUX INP Ensc

IPROG - Projet 2024 JURENSCIC WORLD

Mathis GARNIER - 1ère année - groupe 4

Sommaire :

1. Introduction

1.1. Contexte du projet

Ce projet consiste à développer un jeu sur console inspiré de l'univers Jurassic World où les personnages (Owen, Maisie, Blue et l'Indominus Rex) interagissent entre eux sur un plateau (en mode console, sous VS Code).
L'objectif était de mettre en pratique tout ce que nous avons appris durant un semestre dans le module IPROG du semestre 5.
Le but derrière ce projet était de proposer une expérience ludique et immersive où chaque personnage joue un rôle unique, avec des règles de jeu simples et dynamiques.

Ce jeu nous a permis de mettre de développer nos compétences de développement en C# en mettant en avant des notions comme la manipulation de tableaux, la gestion des entrées utilisateur, et l'affichage interactif sur console.

1.2. Objectif du jeu

Le jeu consiste à éviter que l'Indominus Rex ne mange les personnages humains (Owen et Maisie) tout en essayant de l'isoler sur le plateau à l'aide de grenades. Le joueur doit diriger les personnages Owen et Blue pour atteindre cet objectif en gérant stratégiquement les mouvements aléatoires de Maisie et l'Indominus et ceux choisis pour Owen et Blue.

Ce jeu sert aussi de démonstration technique pour illustrer l'utilisation de concepts clés en programmation :

- Conception d'une matrice 2D comme modèle de plateau de jeu.
- Gestion des interactions et des états des personnages.
- Intégration de tests unitaires pour vérifier les scénarios de jeu.

Ce projet met l'accent sur la lisibilité du code, la modularité et la facilité de test, ce qui le rend très éducatif pour les développeurs débutants comme nous car mettant en avant les différentes normes mises en place en entreprise comme les normes *"PascalCase"* et "*camelCase"* ou bien l'ajout de commentaires afin d'expliquer certaines parties techniques de notre code.

J'ai par ailleurs pu apprendre à implémenter l'énumération "ConsoleKey" qui m'a permis d'accéder aux touches du clavier ce qui était plus pratique que demander à l'utilisateur un chiffre à écrire pour tout ce qui était gestion de déplacement et lancement de grenades.

2. Choix de modélisation des données

2.1. Structure du plateau

Le plateau est modélisé sous la forme d'une matrice 2D de type char[,]. Chaque case représente une position pouvant contenir :

- Un personnage ('O' pour Owen, 'M' pour Maisie, 'B' pour Blue, 'I' pour l'Indominus Rex).
- Une case vide ('.').
- Une grenade ou un éclat d'explosion ('X').

Cette structure permet une représentation claire et efficace du plateau de jeu, facilitant les calculs de position et les interactions entre les entités, j'ai de plus éviter de faire un affichage de la matrice trop complexe car mon code devenait vite compliqué à réaliser avec l'ajout de barre | pour enfermer la structure du plateau. Par ailleurs, les dimensions du plateau sont configurables par l'utilisateur au démarrage grâce à des Console.ReadLine().

2.2. Gestion des personnages

Pour ce qui est de la gestion des personnages, chaque personnage est identifié par un caractère unique dans la matrice. Les déplacements et actions sont gérés via des fonctions spécifiques :

- Owen : Contrôlé directement par le joueur via les flèches directionnelles du clavier.
- Maisie : Se déplace aléatoirement en vérifiant que la case cible est libre.
- Blue : Pousse l'Indominus Rex lorsqu'elle entre en contact avec elle et comme Owen, est contrôlé directement par le joueur via les flèches directionnelles du clavier.
- Indominus Rex : Se déplace aléatoirement et peut manger Owen ou Maisie si elle atteint leur position (donc même coordonnées (x,y)).

Les positions des personnages sont détectées à l'aide de la fonction TrouverPosition() qui retourne leurs coordonnées dans la matrice ce qui facilite le reste du code car nous n'avons pas besoin de trouver à chaque fois pour chaque fonction où se trouve chaque personnage.

Ces mécaniques sont gérées par les fonctions DeplacerOwen() , DeplacerMaisie() , DeplacerBlue(), DeplacerIndominus().

2.3. Mécanique des grenades et des états du jeu

Les grenades sont une ressource limitée (égale à la longueur choisis par le joueur en début de partie) que le joueur peut utiliser pour isoler l'Indominus Rex. Lorsqu'une grenade est lancée, elle impacte une case cible ainsi que les cases adjacentes (gestion des éclats) et marque le plateau de jeu par un X. Les mécaniques principales liées à cette gestion des grenades sont :

- Lancer de grenade : Le joueur choisit une direction et une distance à partir de la position d'Owen.
- Effets des grenades :
    - Marquage de la case cible et des cases affectées par des éclats ('X').
    - Si un personnage est touché, la partie se termine immédiatement.

Les conditions de victoire et de défaites principales sont définies pour garantir un déroulement stratégique et équilibré du jeu mais aussi pour que le jeu ne dure pas éternellement et que le joueur est un objectif :

- L'unique condition de victoire est :
    - L'Indominus Rex est isolée par les grenades, ce qui signifie qu'elle ne peut plus atteindre Owen ni Maisie. Blue n'est pas pris en compte comme condition de défaite, car elle ne risque pas d'être mangée si elle se retrouve dans le même enclos que l'Indominus Rex.
- Les conditions de défaites sont :
    - Un personnage (Owen ou Maisie) est mangé par l'Indominus Rex.
    - Toutes les grenades ont été utilisées sans réussir à isoler l'Indominus Rex.

Ces mécanismes assurent une tension constante tout au long de la partie, obligeant le joueur à faire attention à l'utilisation de ces grenades et à anticiper les mouvements des personnages. Les fonctions IndominusEstIsolee() et LancerGrenade() et son lien avec la fonction VerifierGrenadeTouchePersonnage() jouent un rôle majeur pour valider ces conditions, garantissant ainsi une exécution fluide et des résultats conformes aux règles du jeu.

3. Structuration du code

3.1. Organisation générale

Le code est conçu pour garantir une lisibilité optimale et une modularité efficace. Chaque fonctionnalité est isolée dans une fonction dédiée, ce qui permet une séparation claire entre les différentes parties du jeu. Cette approche facilite la compréhension du code et rend les modifications ou ajouts de nouvelles fonctionnalités plus simples.

- **Affichage** : Les fonctions comme `AfficherPlateau()` et `AfficherTitre()` sont responsables de la présentation visuelle. Elles sont indépendantes des mécanismes du jeu, ce qui permet de les réutiliser ou de les améliorer sans affecter les autres modules.
- **Déplacements et actions** : Chaque personnage a ses propres fonctions dédiées (`DeplacerOwen()`, `DeplacerMaisie()`, `DeplacerBlue()`, `DeplacerIndominus()`) pour gérer leurs déplacements et interactions. Cette séparation garantit une gestion spécifique et claire des comportements de chaque entité.
- **Tests unitaires** : Les mécanismes de tests sont encapsulés dans des fonctions autonomes, comme `TestMangeParIndo()` et `TestRepousserIndominusAvecMaisie()`. Ces tests permettent de valider les comportements attendus dans des scénarios précis, garantissant ainsi la fiabilité des mécaniques de jeu.

Cette organisation modulaire permet une évolutivité accrue. Par exemple, ajouter un nouveau personnage ou modifier une mécanique spécifique n'impacte pas le reste du code. Cela reflète les bonnes pratiques en développement logiciel.

3.2. Fonctionnement du menu principal

Le menu principal agit comme un point d'entrée interactif pour l'utilisateur. Il est structuré pour offrir une expérience intuitive et fluide grâce à une boucle `do-while` qui assure la validation des choix de l'utilisateur. Les trois options principales proposées sont :

1. **Lancer le jeu** :
    - Cette option initialise le plateau et démarre la boucle principale du jeu via la fonction `JeuGame()`.
    - `JeuGame()` orchestre les différentes étapes, de l'initialisation à la vérification des conditions de victoire ou de défaite, en passant par les tours de chaque personnage.
2. **Lancer les tests** :
    - Cette option permet de valider les mécanismes du jeu en exécutant des scénarios spécifiques. Par exemple, `TestMangeParIndo()` simule une situation où l'Indominus Rex mange Owen, tandis que `TestRepousserIndominusAvecMaisie()` vérifie la mécanique de recul de l'Indominus lorsqu'elle est poussée par Blue.

3. **Quitter** :
    - Cette option termine proprement le programme en appelant `Environment.Exit(0)`.

Le menu principal est conçu pour être extensible : de nouvelles options ou fonctionnalités peuvent être ajoutées sans affecter la structure existante.

3.3. Dynamique d'exécution du jeu

Le jeu est structuré autour d'une boucle principale `while`, qui gère les tours successifs des différents personnages et évalue en permanence les conditions de victoire et de défaite. Cette boucle assure une exécution fluide et cohérente, permettant au jeu de progresser jusqu'à ce qu'une condition de fin soit atteinte.

La boucle commence par le tour d'Owen.

```
// Tour d'Owen
Console.Clear();
AfficherTitre("Tour d'Owen");
Console.WriteLine("Voici l'état actuel du plateau :");
AfficherPlateau();
Thread.Sleep(1000);
AfficherApercuDeplacement("Owen");
DeplacerOwen();
AfficherAvecDelaiEtEffacement();

// Gestionnairedes grenades
if (grenadesRestantes > 0)
{
    Console.Clear();
    Console.WriteLine($"Grenades restantes : {grenadesRestantes}");
    Console.WriteLine("Voulez-vous lancer une grenade ? (oui/non)");
    string reponseGrenade = Console.ReadLine()!;
    if (reponseGrenade == "oui")
    {
        Console.Clear();
        AfficherTitre("Lancer de grenade");
        Console.WriteLine("Voici l'état actuel du plateau :");
        AfficherPlateau();
        LancerGrenade();
        grenadesRestantes--;
        AfficherAvecDelaiEtEffacement();

        if (IndominusEstIsolee()) break;

        Console.Clear();
    }
}
else
{
    Console.WriteLine("Aucune grenade restante.");
    Thread.Sleep(1500);
}
```

Pour le tour d'Owen (image ci-dessus), l'exécution commence par une phase d'initialisation. L'écran est nettoyé avec `Console.Clear()`, puis le titre du tour est affiché grâce à la fonction `AfficherTitre("Tour d'Owen")`. L'état actuel du plateau est présenté via `AfficherPlateau()`, suivi d'une courte pause introduite par `Thread.Sleep(1000)` pour permettre au joueur de prendre connaissance des informations affichées. Owen peut ensuite se déplacer à l'aide des flèches directionnelles, son déplacement étant géré par la fonction `DeplacerOwen()`. Des animations ou des effets visuels accompagnent cette action, assurés par les fonctions `AfficherApercuDeplacement("Owen")` et `AfficherAvecDelaiEffacement()`.

Le tour d'Owen inclut également une gestion des grenades. Si le joueur dispose de grenades (`grenadesRestantes > 0`), une invitation à en lancer une est présentée. Le nombre de grenades restantes est affiché, suivi d'une question posée via `Console.WriteLine` : *"Voulez-vous lancer une grenade ? (oui/non)"*. Si le joueur répond par l'affirmative, l'écran est à nouveau nettoyé, et le titre *"Lancer de grenade"* est affiché. L'état du plateau est actualisé, et la fonction `LancerGrenade()` est appelée pour exécuter l'action. Une grenade est consommée (`grenadesRestantes--`), et les effets visuels sont mis à jour pour refléter l'impact. Si, après le lancer, l'Indominus Rex est isolée, la boucle principale est interrompue via un `break`. Dans le cas où le joueur choisit de ne pas utiliser de grenade ou si aucune n'est disponible, un message indiquant *"Aucune grenade restante."* est affiché, suivi d'une pause gérée par `Thread.Sleep(1500)`.

Ce code structure le tour d'Owen de manière claire et modulaire. L'affichage initial informe le joueur sur l'état du jeu, tandis que la gestion des actions (déplacement et lancer de grenade) est isolée dans des blocs logiques distincts. Les effets visuels et les pauses renforcent l'immersion et rendent l'expérience plus fluide. Cette organisation rend le code facile à maintenir et à étendre, tout en garantissant une expérience utilisateur interactive et engageante.

Des mécanismes similaires sont mis en place pour les autres personnages, chacun ayant des caractéristiques propres :

- **Maisie** se déplace de manière aléatoire via la fonction `DeplacerMaisie()`, qui vérifie la disponibilité des cases adjacentes. Si elle est bloquée, son tour est ignoré, ajoutant une dimension stratégique au jeu.
- **Blue** peut se déplacer ou repousser l'Indominus Rex si elle entre en contact avec elle. La mécanique de recul est gérée par la fonction `RepousserIndominus()`, qui prend en compte les limites du plateau et les obstacles.
- **L'Indominus Rex** se déplace de manière aléatoire grâce à la fonction `DeplacerIndominus(out bool personnageMange)`. Si elle atteint Owen ou Maisie, la partie se termine immédiatement, remplissant une condition de défaite.

Enfin, la boucle principale `while` encapsule toutes ces dynamiques, vérifiant après chaque tour si une condition de victoire (par exemple, l'isolement de l'Indominus Rex) ou de défaite (capture d'Owen ou de Maisie) est remplie. Cette structure garantit une progression logique et immersive tout en maintenant une interaction constante avec le joueur.

4.Planning de réalisation

4.1. Phases de développement

Pour la création de ce jeu, plusieurs phases ont été nécessaires.

La première étape a consisté en une analyse approfondie de la tâche demandée. Il s'agissait de comprendre les règles du jeu à réaliser ainsi que les différentes contraintes techniques associées. J'ai réfléchi à la manière d'organiser le programme et décidé de structurer le jeu de manière classique, avec des tours successifs pour chaque personnage. Chaque personnage exécute ses actions dans un ordre prédéfini (Owen, Maisie, Blue, puis l'Indominus). Ce schéma se répète jusqu'à ce qu'une condition de victoire ou de défaite soit remplie.

Dans cette phase initiale, j'ai posé les bases de l'élaboration du jeu en créant les différentes fonctions nécessaires, sans les remplir immédiatement. Cela m'a permis de structurer l'architecture globale du code en mettant en avant la modularité, la gestion des fonctions, et une organisation claire en modules.

Ensuite, j'ai commencé à implémenter les fonctionnalités fondamentales du jeu, en me concentrant d'abord sur l'affichage du plateau et le déplacement des personnages. Une fois ces éléments de base en place, j'ai introduit la gestion des tours grâce à une boucle `while`. À ce stade, le jeu se limitait à un plateau contenant quatre personnages, dont deux étaient contrôlés par l'utilisateur à l'aide de commandes `Console.ReadLine`. Cependant, pour une expérience plus pratique et immersive, j'ai ensuite opté pour un contrôle via les touches du clavier.

Le code était encore basique à ce moment-là, et j'ai donc ajouté progressivement des mécaniques avancées. L'une des parties les plus complexes à coder a été la gestion des lancers de grenades, associée à la condition de victoire. Après de nombreuses heures de réflexion et de tests, j'ai implémenté les conditions de victoire et de défaite. Ces étapes ont nécessité l'utilisation de fonctions de test pour valider chaque mécanique, comme `TestMangeParIndo()` et `TestRepousserIndominusAvecMaisie()`. Bien que plusieurs autres tests aient été effectués pour vérifier les différentes conditions, j'ai supprimé certains d'entre eux au fil du temps pour alléger le code. Les dernières fonctions testées incluent `RepousserIndo()` et la condition de défaite où l'Indominus attrape Owen ou Maisie.

Une fois les mécaniques avancées en place, j'ai ajouté des effets visuels et des animations pour améliorer l'expérience utilisateur et rendre le jeu plus immersif.

La phase finale du développement a été consacrée au débogage. Certains bugs ont été corrigés, notamment dans la logique des déplacements et des interactions, comme avec la fonction `RepousserIndominus()`, qui posait problème lorsque l'Indominus traversait des obstacles ou d'autres personnages. Après avoir résolu ces problèmes, j'ai nettoyé le code pour améliorer sa lisibilité et ajouté des commentaires pour faciliter sa compréhension. Ces dernières étapes ont marqué la fin du développement de mon projet.

4.2. Gestion des imprévus

L'un des premiers imprévus concernait la logique des déplacements. Par exemple, la fonction `RepousserIndominus()` a rencontré des problèmes inattendus, notamment lorsque l'Indominus traversait des obstacles ou des personnages après avoir été repoussée. Il a également été difficile de comprendre précisément depuis quelle position Blue devait pousser l'Indominus pour que le mouvement reste cohérent avec les règles du jeu et les limites du plateau. Ces erreurs ont nécessité une révision approfondie du code, avec plusieurs tests et ajustements pour obtenir un résultat satisfaisant.

Un autre imprévu majeur est survenu lors de l'implémentation des lancers de grenades. La gestion des impacts et des éclats sur le plateau s'est avérée plus complexe que prévu, notamment pour déterminer si l'Indominus était isolée ou non après l'explosion. Cela m'a obligé à repenser certaines parties du code et à effectuer de nombreux tests pour valider le fonctionnement de cette mécanique.

La transition des contrôles des personnages de `Console.ReadLine` vers des touches clavier a également été un défi inattendu. Bien que cela ait considérablement amélioré l'expérience utilisateur, cette modification a nécessité des recherches supplémentaires pour implémenter correctement les événements clavier dans un environnement console, ce qui n'était pas prévu initialement.

Enfin, la gestion des conditions de victoire et de défaite a présenté des imprévus liés à des cas particuliers. Par exemple, il a fallu gérer les scénarios où l'Indominus était isolée mais pouvait encore se déplacer vers des personnages, ou encore des cas où Maisie était bloquée par des obstacles, empêchant la progression normale du jeu et risquant de créer une boucle infinie. Ces situations ont nécessité l'ajout de vérifications supplémentaires et de mécanismes pour éviter les blocages ou les comportements incohérents.

Pour surmonter ces imprévus, j'ai adopté une approche itérative : chaque problème a été isolé, analysé, puis corrigé à l'aide de fonctions de test spécifiques. J'ai également accordé une attention particulière à la modularité du code, ce qui m'a permis de modifier ou d'améliorer certaines parties

sans affecter l’ensemble du programme. Malgré ces imprévus, ces ajustements ont enrichi le processus de développement et permis d’aboutir à un jeu fonctionnel et cohérent.

5. Bilan

5.1. Points forts du projet

Pour parler maintenant des points forts de mon projet, j’ai structuré mon code de manière modulaire et lisible. Chaque action ou comportement du jeu comme les déplacements des personnages ou le lancer de grenades est encapsulé dans des fonctions spécifiques. J’ai aussi essayé d’être rigoureux sur les conventions de nommage comme PascalCase et camelCase et en ajoutant des commentaires explicatifs clairs.

J’ai respecté au maximum ce qui était demandé en terme de contraintes techniques et de fonctionnalités à implémenter

Un autre aspect sur lequel j’ai insisté est la validation rigoureuse des mécanismes du jeu. J’ai intégré des tests pour m’assurer du bon fonctionnement des déplacements, des interactions entre personnages, et de certaines mécaniques comme le recul de l’Indominus Rex ou la gestion des grenades. Ces tests m’ont permis de garantir que chaque partie critique du code fonctionne correctement et selon mes attentes.

Enfin, ce projet a été pour moi une excellente opportunité d’apprentissage. J’ai découvert et mis en œuvre des concepts avancés en programmation comme l’utilisation d’une matrice en 2D comme modèle de plateau. Ce développement m’a également familiarisé avec des pratiques professionnelles, telles que la modularité et le respect des normes de codage, qui me seront précieuses dans des projets futurs.

5.2. Axes d'amélioration

Pour ce qui est des axes d’amélioration du jeu que j’aurai pu effectué, j’aurai pu par exemple améliorer l’expérience, les temps d’attente que j’ai introduits avec Thread.Sleep, bien qu’ils ajoutent un certain rythme, peuvent paraître long et nuire à la fluidité du jeu.
De plus, ajouter un tutoriel ou améliorer les explications des commandes et des règles dans le menu principal pourraient être un plus pour faciliter la prise en main, en particulier pour les joueurs découvrant le jeu pour la première fois.

Un autre axe d'amélioration concerne l'optimisation du code. Certaines fonctions, comme LancerGrenade() ou DeplacerIndominus() sont relativement longues et complexes. Je pourrais les décomposer davantage pour améliorer leur lisibilité.
Les fonctions de déplacement étant relativement similaires comme DeplacerBlue() et DeplacerOwen() auraient pu être regroupées en une seule fonction.

En ce qui concerne les tests, j'ai implémenté très tard une fonctionnalité qui me permettrait de voir si mes fonctions faisaient ce que je t'attendais d'elles. C'est un point auquel je penserai car que j'ai identifié comme essentiel pour la bonne continuité de mon projet.

De plus, j'aurais pu ajouter certaines fonctionnalités comme des ajouts de personnages ou autres.

Enfin, l'esthétique du jeu sur console pourrait être améliorée. Actuellement, bien que le plateau soit fonctionnel, il manque de clarté visuelle. Ajouter un cadre autour du plateau ou des bordures pour mieux délimiter les zones améliorerait sa lisibilité..

Ces améliorations auraient été ce que j'aurais pu rajouter et pourraient considérablement enrichir l'expérience globale et rendre le projet plus abouti.

5.3. Résultats obtenus

Je suis globalement satisfait des résultats obtenus, tant sur le plan technique que pédagogique. Le jeu est fonctionnel et respecte les objectifs fixés au départ. Les conditions de victoire et de défaite sont correctement implémentées et les mécaniques de jeu, telles que la gestion des déplacements, des grenades et des interactions entre personnages s'exécutent sans problème majeur (où du moins, je n'en ai pas repéré).

Sur le plan technique, j'ai réussi à développer un jeu console qui exploite des concepts avancés, comme l'utilisation d'une matrice 2D pour modéliser le plateau. Ce travail m'a permis de consolider mes compétences en programmation en C# et d'approfondir ma compréhension des structures de données et de la gestion des interactions dans un environnement interactif donc j'en suis relativement satisfait.

6. Annexes

6.1. Diagramme de structure du plateau

Le plateau du jeu est modélisé sous la forme d'une matrice bidimensionnelle (`char[,]`) où chaque élément représente une case ayant une signification spécifique :

- `'O'` : Représente le personnage Owen.
- `'B'` : Représente le personnage Blue.
- `'M'` : Représente le personnage Maisie.
- `'I'` : Représente l'Indominus Rex.
- `'X'` : Indique une case marquée par une grenade ou un éclat.
- `'.'` : Représente une case vide.

Le plateau est initialisé en fonction des dimensions choisies par l'utilisateur. Chaque personnage est positionné aléatoirement sur une case libre au démarrage. Cette structure simplifie la gestion des interactions entre les entités du jeu.

6.2. Diagramme de séquence du jeu

Voici une description textuelle du déroulement typique d'une partie sous forme de séquence :

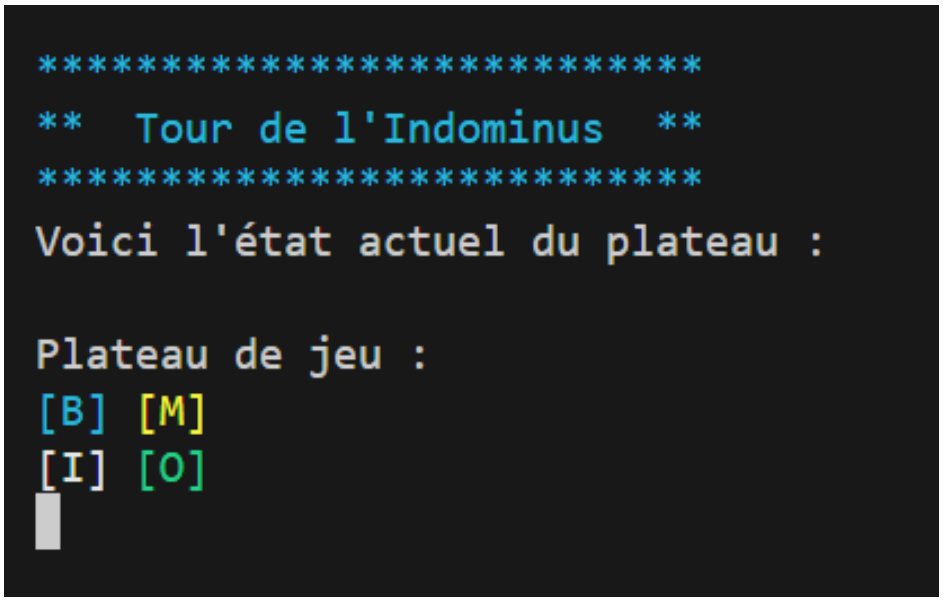
1. **Menu principal :** L'utilisateur choisit entre lancer le jeu, exécuter des tests ou quitter le programme.
2. **Initialisation :** Le plateau est généré, les personnages sont positionnés, et le joueur configure les dimensions et le nombre de grenades.
3. Tour de jeu :
    - **Owen :** Le joueur déplace Owen à l'aide des flèches directionnelles. S'il reste des grenades, il peut choisir d'en lancer une, ce qui impacte le plateau.
    - **Maisie :** Se déplace de manière aléatoire vers une case adjacente libre.
    - **Blue :** Le joueur déplace Blue de manière similaire à Owen. Si elle entre en contact avec l'Indominus, elle la repousse.
    - **Indominus Rex :** Se déplace aléatoirement et peut manger Owen ou Maisie si elle les atteint.
4. **Vérification des conditions de fin :** À chaque tour, les conditions de victoire (isolement de l'Indominus Rex) ou de défaite (Owen ou Maisie mangé, plus de grenades disponibles) sont évaluées. Si une condition est remplie, la partie se termine.

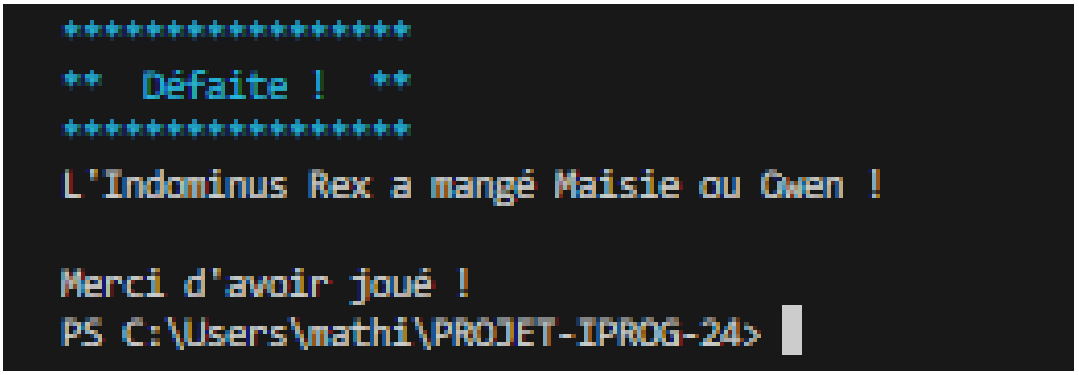
Ce cycle se répète jusqu'à ce qu'une condition de fin soit atteinte.

6.3. Exemples de tests et résultats

Test 1 : L'Indominus Rex mange Owen

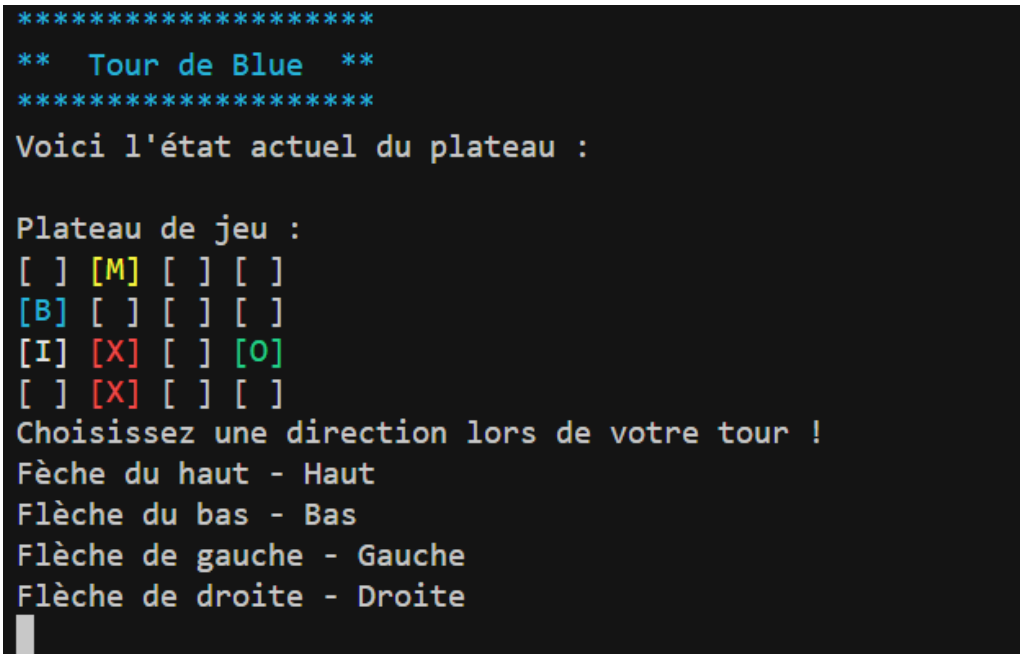
- **Scénario :** L'Indominus Rex est placée à côté d'Owen sur le plateau. On simule un tour de l'Indominus où elle doit se déplacer vers Owen.
- **Résultat attendu :** L'Indominus atteint la position d'Owen, et le booléen personnageMange devient true.
- **Résultat obtenu :** Lors de l'exécution, l'Indominus a correctement mangé Owen, et la condition de défaite s'est activée.

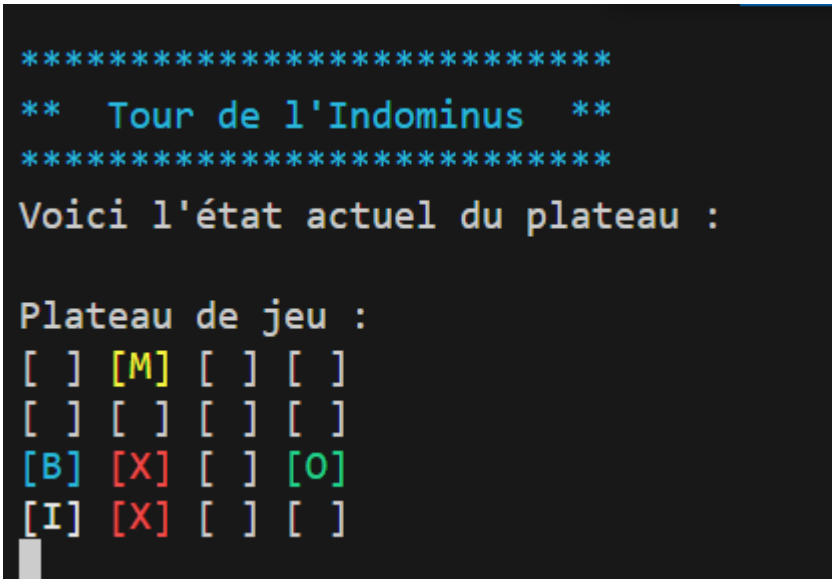




Test 2 : Recul de l'Indominus Rex lorsque Blue la pousse

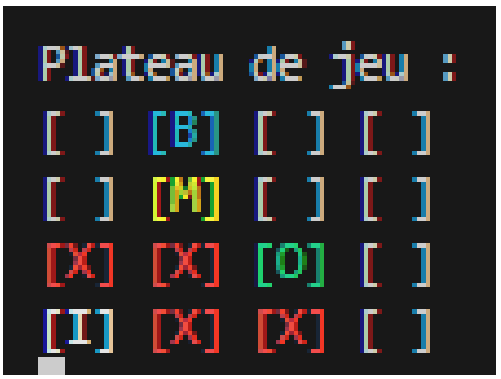
- **Scénario :** Blue est positionnée à côté de l'Indominus Rex. On simule un déplacement de Blue pour la repousser.
- **Résultat attendu :** L'Indominus recule de trois cases ou jusqu'à ce qu'elle rencontre un obstacle.
- **Résultat obtenu :** L'Indominus a bien reculé de manière cohérente selon les règles définies.

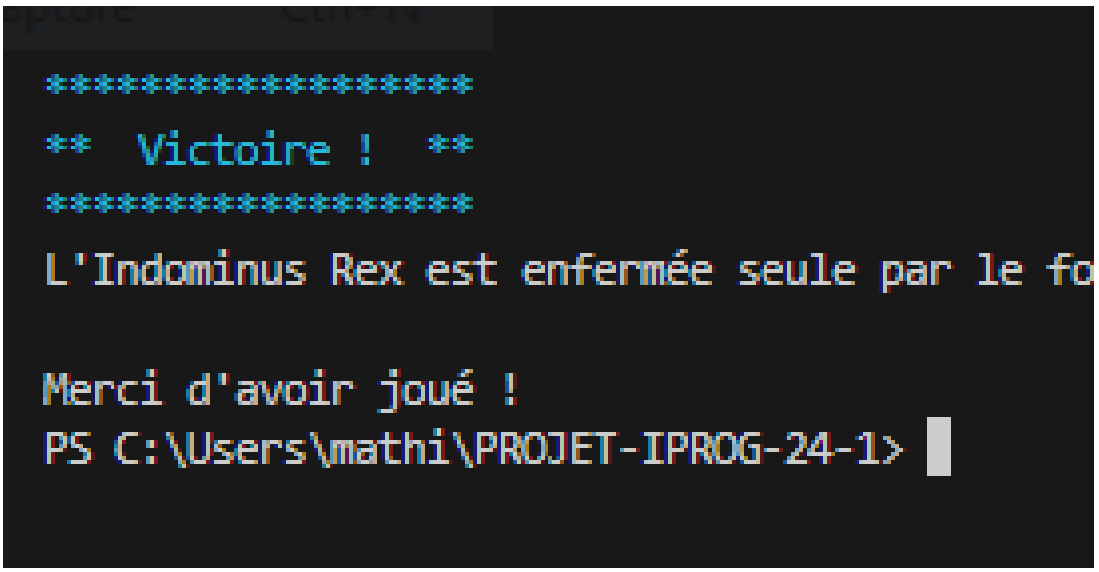




Test 3 : Condition de victoire

- **Scénario :**
  L'Indominus Rex est placée sur le plateau et entourée de crevasses (`'X'`) ou des bords du plateau. On vérifie si la fonction `IndominusEstIsolee()` détecte correctement qu'elle est isolée.
- **Résultat attendu :**
  La fonction retourne `true`, et un message de victoire est affiché pour signaler que l'Indominus ne peut plus atteindre Owen ni Maisie.
- **Résultat obtenu :**
  Lors de l'exécution, la fonction a correctement détecté que l'Indominus Rex était isolée, et la condition de victoire a été activée.





7. Conclusion

Ce projet m’a permis de consolider mes compétences en programmation, notamment en C#, tout en abordant des concepts avancés comme la gestion d’une matrice en 2D et l’interaction en temps réel. La réalisation de ce jeu m’a aussi familiarisé avec des pratiques professionnelles telles que la modularité du code et la validation des fonctionnalités par des tests.

Bien que le jeu soit fonctionnel et respecte les objectifs initiaux, des axes d’amélioration comme une meilleure fluidité ou une interface plus claire pourraient encore enrichir l’expérience utilisateur. Ce travail représente une étape significative dans mon apprentissage, et les compétences acquises me seront utiles pour mes futurs projets techniques notamment en ce qui concerne le développement en C# et l’utilisation de GitHub.