CodeTris

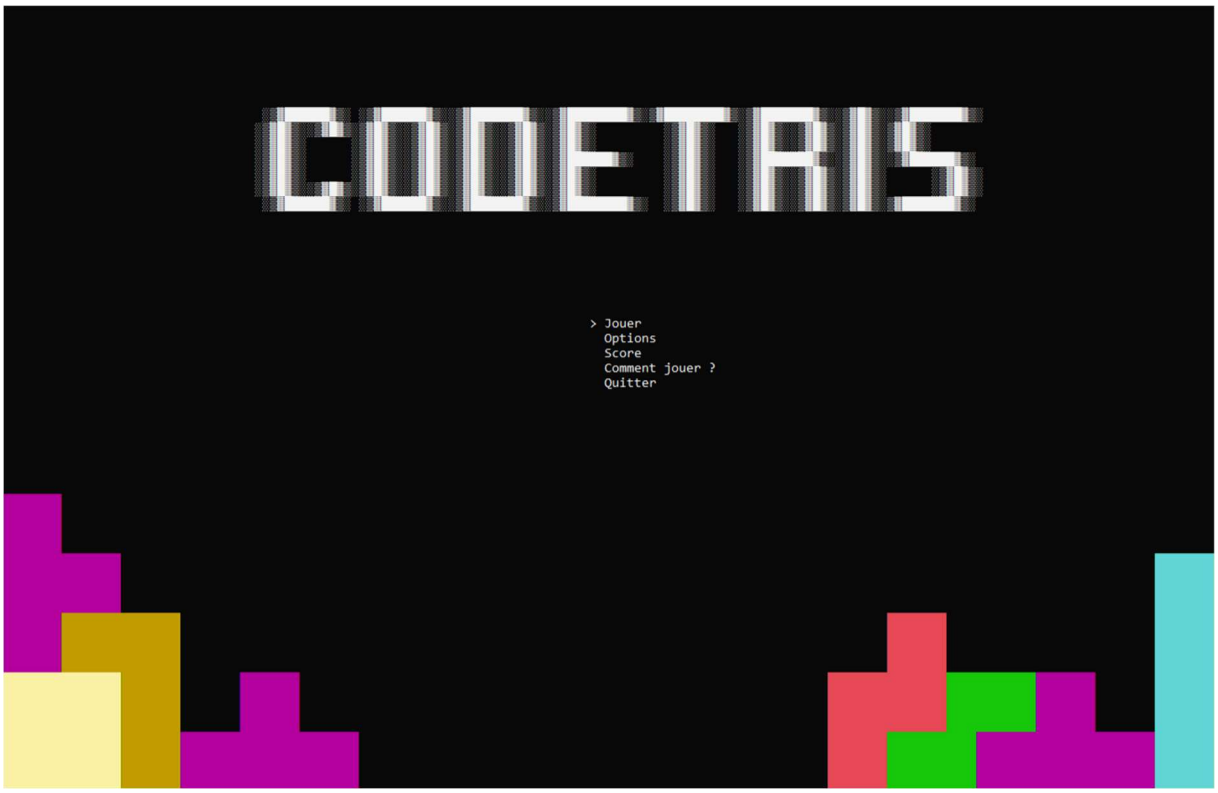

Figure 1 : Image représentant le projet

Alexandre King – MID4
Avenue de Valmont 28b, 1010 Lausanne
88h
Aurélie Curchod

Table des matières

1 RÉSUMÉ

Ce document permet de comprendre comment réaliser l'application CodeTris (Tetris combiné avec des questions de C#) en C# console ainsi que de reproduire celle-ci. Les outils utilisés au départ sont Visual Studio 2022 ainsi qu'un serveur UWamp pour la BD.
Le but de CodeTris, est de pouvoir jouer à Tetris tout en complétant des questions sur le C#. Lorsqu'une ligne se bloque, une question est posée (donc si le joueur complète 4 lignes en même temps, 4 questions lui seront posés). Le public cible pourrait être des élèves de première année en informatique afin que ceux-ci puissent apprendre les bases de C# de manière ludique. Une planification initiale sera mise en œuvre au début du projet afin d'assurer au mieux le déroulement de celui-ci et d'arriver à le finir dans un temps imparti de 88 heures (documentation comprise).

Le projet sera réalisé en C# console avec comme template .Framework afin de profiter de certains aspects non-présent dans .NET. L'utilisation des classes permettra de bien segmenter le code afin de mieux suivre la logique du programme et le rendre plus « propre » et lisible. Certaines classes feront partie de la famille des managers. Elles seront les classes maitresses de certains points importants comme le jeu (Tetris), le menu ou bien encore la base de données. D'autres classes feront, quant à elles, parties de la famille des classes statiques. L'utilisation de classes statiques permet de ne pas avoir les instanciées. Ces classes sont souvent appelées à beaucoup d'endroit dans le code ou non pas nécessairement besoin d'être instancié.
Les tetriminos seront gérés grâce à un tableau bidimensionnel représentant la grille de jeu (celle-ci étant plus grande visuellement, une grille avec les « vrais » dimensions est nécessaire). Pour les pièces en elles-mêmes, une classe Tetriminos sera parent de classes enfants pour chaque pièces (7 au total).
Les éléments seront abordés dans l'ordre suivant concernant la réalisation : Le menu, les options, la base de données, l'interface du jeu, la gestion des Tetriminos, les questions de C#, le game over, le score et enfin, la mise en base de données des parties. L'ordre suit la logique d'une partie lambda d'un joueur. Il commence le jeu dans le menu, (choisit les options pour sa partie,) lance le jeu, joue avec les Tetriminos, répond aux questions sur le C#, perd la partie, son score est alors enregistré à ce moment-là, le jeu lui affiche son score final et son nombre de bonnes et de mauvaises réponses, il peut alors retourner au menu principal.

Après 88 heures de projet, celui-ci est fini à 95%. Il ne reste que quelques petits bugs venant des collisions et des rotations. Dans l'ensemble, le projet s'est bien déroulé. Cela est peut-être dû à la planification qui prévoyait des marges car il faut souvent débugger ou corriger des bugs. Le temps de marge qui n'a pas été utilisé a permis d'ajouter quelques améliorations et d'optimiser le programme (celui-ci prenant entre <1% et 2% du processeur environ). Grâce a une bonne logique, les tetriminos ont pu être gérer de manière optimal expliquant aussi le fonctionnement du programme avec un faible coût de ressources. Concernant la documentation, celle-ci a été mise un peu de côté lors de la création du programme. Des notes y étaient régulièrement ajoutées mais la mise au propre ne s'est faite qu'après (du moins pour la réalisation, l'analyse et conception ont été faites en parallèle du programme). Ce qui pourrait encore être fait actuellement dans le programme serait de corriger les bugs restants, faire un vrai menu pause et ajouter des questions en DB pour que le jeu ne soit pas trop répétitif (environ 30 questions par difficulté minimum pour avoir un bon jeu). Pour la documentation, des diagrammes UML pourraient être ajoutés comme un diagramme de classes par exemple.

2 SPÉCIFICATIONS

2.1 Titre

CodeTris : Le bloc de la programmation

2.2 Description

CodeTris est un jeu éducatif qui intègre des questions de programmation dans le jeu d'arcade Tetris.

L'objectif de ce projet est de créer une expérience ludique et éducative, encourageant les joueurs à renforcer leurs compétences en programmation tout en jouant à un jeu classique.

Ce jeu pourrait être utilisé par exemple comme accroche lors des modules de programmation de 1ère année.

2.3 Matériel et logiciels à disposition

- PC de l'ETML
- Visual Studio 2022
- Github
- Suite office

2.4 Cahier des charges

Voir annexes

2.5 Les points suivants seront évalués

- Le rapport
- Les planifications (initiale et détaillée)
- Le journal de travail
- Le code et les commentaires
- Les documentations de mise en œuvre et d'utilisation

2.6 Validation et conditions de réussite

- Compréhension du travail
- Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
- Etat de fonctionnement du produit livré

3 PLANIFICATION INITIALE

Cette partie montre la planification initiale du projet, elle montrera comment le temps a été réparti et quels sont les paliers critiques. Elle indiquera également le début et la fin du projet ainsi que les jours fériés.

3.1 Répartition du temps

Cette partie montre la répartition du temps en heure et en pourcent ainsi que la répartition selon le cahier des charges afin de comparer.

Partie du projet concerné	Temps planifié (En heure et en pourcent)	Temps recommandé selon le CDC
Analyse	5.42 heures 6.15%	17.6 heures soit 20%
Réalisation	43.33 heures soit 49.23%	39.6 heures soit 45%
Documentation	35.26 heures soit 40.06 %	22 heures soit 25%
Tests	5 heures soit 5.68 %	8.8 heures soit 10%

3.2 Dates importantes

Le projet a débuté le 29 avril 2024 et il se finit le 29 mai 2024 (pour des raisons d'absences lors du projet, la date de fin a été déplacé au 30 mai 2024).

Les jours suivant ne sont comptés lors de la réalisation du projet :

- Tous les mardis et tous les jeudis matin (cours de MATU)
- Jeudi 9 mai et vendredi 10 mai (Ascension)
- Lundi 20 mai (Pentecôte)

3.3 Planification dans Excel

Pour des raisons de mise en page et de lisibilité, le diagramme de Gantt fait dans Excel se trouve dans les annexes.
// spécifié la page Une fois le rapport fait ?

3.4 Méthode de gestion de projet utilisée

La méthode de gestion de projet utilisée est la méthode des 6 pas.
Elle consiste en :

1. Informer (s'informer sur le projet)
2. Planifier (planifier en fonction du temps et des moyens/demandes du client)
3. Décider (choisir comment sera fait le projet (sélection du template))
4. Réaliser (réaliser le projet)
5. Contrôler (vérifier son bon fonctionnement avec des tests)
6. Évaluer (évaluer le résultat (fait par les experts et la cheffe de projet))

3.5 Modification de la planification

Comme précisé au point 3.2, dû à une absence, le délai a été repoussé au jeudi 30 mai. Il faut donc compter un décalage d'une demi-journée à partir du mercredi 1 mai.

4 ANALYSE

4.1 Opportunités

- Mise en œuvre d'un jeu avec C# console
- Utilisation de GitHub
- Jeu éducatif pour des élèves de première année

4.2 Objectif

Le but du projet est de réaliser un jeu mêlant tetris et le C#. Il pourrait être présenté à des élèves de première année dans le but de leur faire apprendre le C# de manière ludique.

4.3 Document d'analyse et conception

4.3.1 Analyse de l'application

Une analyse de l'application serait faite dans un premier temps. Ensuite, la conception sera abordée au point 4.3.2

Chaque élément important aura droit à sa section lors de cette analyse.

4.3.1.1 Général

Les éléments principaux ne faisant pas parti d'une section spécifique seront abordés dans ce chapitre.

Le template utilisé sera C# console .Framework. Dans le cas où de la musique serait ajoutée plus tard dans l'application (ajout venant du point 4.3.1.5), il sera nécessaire de pouvoir utiliser le using System.Media permettant d'utiliser les SoundPlayer. Ceux-ci n'étant pas disponible avec C# console .NET, il faut passer par un moyen bien plus long pour pouvoir ajouter du son.

4.3.1.2 Menus

Le menu principal doit contenir les éléments suivants :

Nom du choix	Description
Jouer	Permet de lancer le jeu
Options	Affiche le menu des options
Meilleurs scores	Permet au joueur de voir les meilleurs scores en fonction de la difficulté
Tuto / aide pour le joueur	Tuto expliquant brièvement le jeu (doc utilisateur intégrée)
Quitter	Quitte l'application

Le menu des options doit contenir les éléments suivants :

Nom du choix	Description
Touches	Permet de choisir si l'on veut jouer avec les

	touches WASD ou les flèches
Difficulté	Change la difficulté
Retour	Revient au menu principal

Ayant déjà prévu d'ajouter du son, une option « musique » sera également ajoutée. Elle permettra de couper ou lancer la musique.

Concernant le menu des meilleurs scores, celui-ci sera sous forme de sous-menu où l'on pourra choisir la difficulté pour laquelle on souhaite voir les scores. Un score est représenté par un nom de joueur (pseudo), un nombre de points et la date du record.

Le déplacement dans le menu se fera via les flèches. Ce symbole « > » permettra de voir quel sous-menu/option est présélectionné et la touche ENTER permet d'afficher notre sélection.

Dans le menu des options, le mot « musique » passera de rouge à vert selon si elle est activée ou non.

Pour les touches, il sera écrit à côté du texte « touches : » quelles sont les touches actuellement actives (« WASD » ou « Flèches »).

Concernant la difficulté, il sera écrit à côté du texte « difficulté : » quelle est la difficulté actuelle. De plus, une couleur sera attribuée à chacune d'entre elles (vert = facile, jaune foncé = moyen, rouge = difficile).

Le menu principal bénéficiera également d'éléments visuels ''décoratifs'' comme un titre en ascii et des Tetriminos géants afin de directement reconnaitre le jeu au lancement.

4.3.1.3 Jeu

Un tetris fera office de base du jeu. Il faudra donc y intégrer les éléments suivants :

- Une grille où les pièces tomberont
- Des tetriminos (ceux du jeu de base avec leur couleur)
- La possibilité de faire pivoter les tetriminos
- La possibilité de les déplacer sans qu'ils ne sortent de la zone dédiée
- Ils devront s'empiler
- Ils doivent pouvoir compléter une ligne qui disparaitra par la suite
- Des lignes doivent pouvoir se bloquer pour le reste de la partie

Lorsqu'une ligne sera complétée, une question sur le C# (console et/ou forms) sera posée. Le nombre de questions correspond au nombre de lignes pleines (Exemple, si le joueur rempli 1 ligne, alors 1 question lui est posée. S'il remplit 3 lignes, 3 questions lui sont posées. Le nombre maximum de question est donc de 4 car au maximum, seulement 4 lignes peuvent être complétés à la fois).
Les questions seront stockées en base de données et une difficulté leur sera attribuée. Plus une question est difficile, plus elle rapporte de points.

Si le joueur répond correctement à la question, des points lui sont attribués, en revanche, s'il répond mal, aucun point ne lui seront attribué et il subira en plus un malus. Pour l'instant, le seul malus demandé est de bloquer une ligne, réduisant ainsi la surface de jeu. En cas de mauvaise réponse, la réponse attendue s'affiche sur l'écran et un signal visuel averti le joueur. Lorsque le joueur atteint le haut du niveau avec un tetriminos, la partie s'arrête déclenchant le ''game over''. Le joueur voit alors son score ainsi que son nombre de réponses correctes et incorrectes. Il peut ensuite retourner au menu principal pour relancer une partie.

4.3.1.4 Base de données

La base de données devra stocker les éléments suivants :
- Les utilisateurs
- Les difficultés
- Les questions
- Les parties des utilisateurs

On fera d'abord un MCD afin de conceptualiser la base de données. Le MCD et MLD seront abordés dans la conception

Dû au fait qu'une base de données sera présente, il faut également penser à ajouter un fichier de logs en cas de problème avec celle-ci. Pour éviter que le programme ne crash en cas de problème avec la base de données, un try catch sera mis en place. Le catch récupérera l'erreur et l'inscrira dans le fichier de logs. Il faudra donc gérer des documents extérieurs au programme grâce au StreamWriter et StreamReader et au using System.IO.

4.3.1.5 Améliorations possibles

Il est déjà possible d'imaginer plusieurs améliorations au programme. En voici quelques-unes :
- Ajout de musique dans les menus et le jeu
- Possibilité de choisir son pseudo ou de le modifié
- En jeu, une zone dédiée montre le prochain tetriminos
- Ajout de difficulté en plus
- Ajout de malus en cas de mauvaise réponse à une question
- Un menu « pause » lorsque l'on est en jeu
- La forme des pièces
- Le choix de la technologie pour les questions (par exemple : HTML+CSS)

4.3.2 Conception de l'application

4.3.2.1 Menus

Voici des maquettes de l'interface des menus :

Figure 2 : Maquette du menu principal

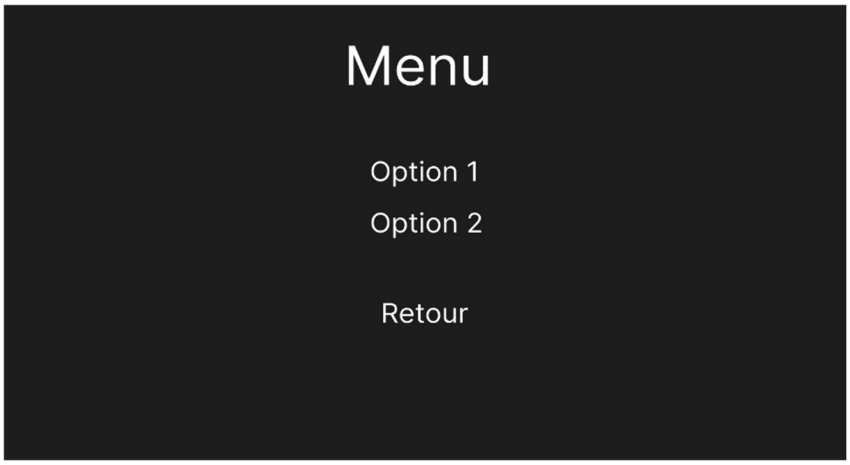

Figure 3 : Maquette des autres menus

Chaque option du menu sera affichée comme une liste que l'utilisateur pourra parcourir avec les flèches. Ce signe ''>'' montrera sur quelle option l'utilisateur se trouve actuellement. Il validera le choix avec la touche ENTER. Pour les sous-menus, une option pour revenir en arrière sera présente. Pour gérer les choix des menus, l'utilisation d'un dictionnaire est pertinente afin d'avoir une clé unique pour les choix associé à un string (Dictionnaire avec un int et un string).

Une boucle Do While combinée avec un Console.ReadKey() permettra de récupérer l'input du joueur à chaque boucle et d'agir en conséquence. Le joueur ne pourra que se déplacer avec les flèches et la touche ENTER. En cas d'une autre entrée, il faudra la gérer pour éviter que le programme fasse autre chose.
La touche sera ensuite récupérée pour décider de la prochaine action que le programme devra exécuter.

4.3.2.2 Jeu

Voici une maquette de l'interface du jeu :

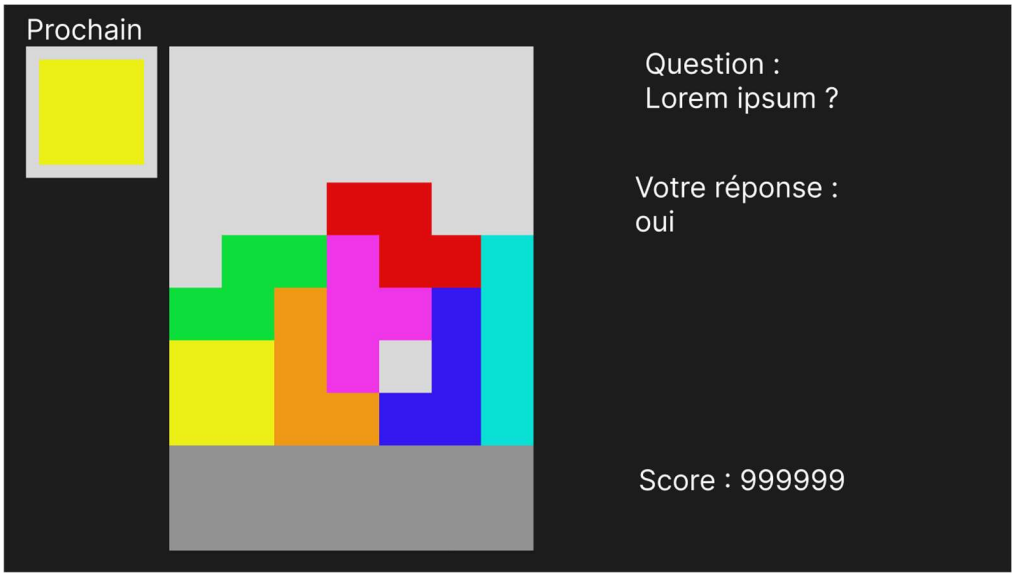

Figure 4 : Maquette du jeu

Les tetriminos y sont un peu grands mais la maquette permet de donner une idée de ceux-ci. La taille de ceux-ci sera définie lors de la réalisation. La maquette prévoit déjà une amélioration, celle de voir la prochaine pièce.
À droite, les questions apparaitront et le joueur pourra écrire sa réponse dessous.

Pour le fonctionnement du jeu, un game manager s'occupera de celui-ci. Il gérera les pièces, les questions et le score (d'autres classes seront surement créée afin de ne pas tout mettre dans le game manager). Les tetriminos utiliseront de l'héritage au vu du fait qu'ils auront tous les mêmes variables et les mêmes méthodes.

Pour la gestion de la zone de jeu, l'arrière-plan (background) sera mis en gris clair et les pièces seront avec du texte (permet de ne pas avoir à réécrire l'arrière-plan à chaque fois). Un tableau de boolean à 2 dimensions va permettre de savoir si une case est occupée ou non. On pourra ensuite vérifier sur la pièce peut se déplacer à un endroit ou non ou bien s'il a touché le bas et qu'il doit se figer.
À chaque fois qu'un tetriminos se fige, on vérifiera si une ligne est complète ou non. Si oui, une question sera posée au joueur. Pour éviter que la même question passe en boucle, il faudra utiliser 2 listes. Une avec les questions possibles et une autre avec celles utilisées. Une fois qu'une question a été posée, elle est retirée de la liste des questions possibles et ajoutée à celle des questions utilisées. Une fois la liste de questions possibles vide, on la rerempli et vide celle des questions utilisées.

Afin de vérifier si une pièce peut tourner, il va falloir sa rotation afin de vérifier sa rotation. Si une simulant la rotation, la pièce et bouger, on tourne la pièce. Si ce n'est pas possible, on laisse la pièce dans son état actuel.

L'utilisation de thread ne sera pas nécessaire. Il sera possible d'exécuter le code de manière séquentielle. Cela aidera également pour l'optimisation du programme.

Au début de chaque partie, il faudra réinitialiser plusieurs variables afin que le jeu puisse fonctionner correctement. Une fonction Reset() permettra de faire cela.
Il faudra également qu'en début de partie, plusieurs variables soient définies en fonction des paramètres de l'utilisateur. Pour ce faire, une ou plusieurs méthodes (en fonction du paramètre par exemple) seront définies pour donner aux variables la bonne valeur.

Le curseur sera caché afin d'améliorer un peu le visuel de l'application et de ne pas perturber le joueur par son grand nombre de déplacements.

4.3.2.3 Base de données

Voici le MCD et MLD de la base de données :

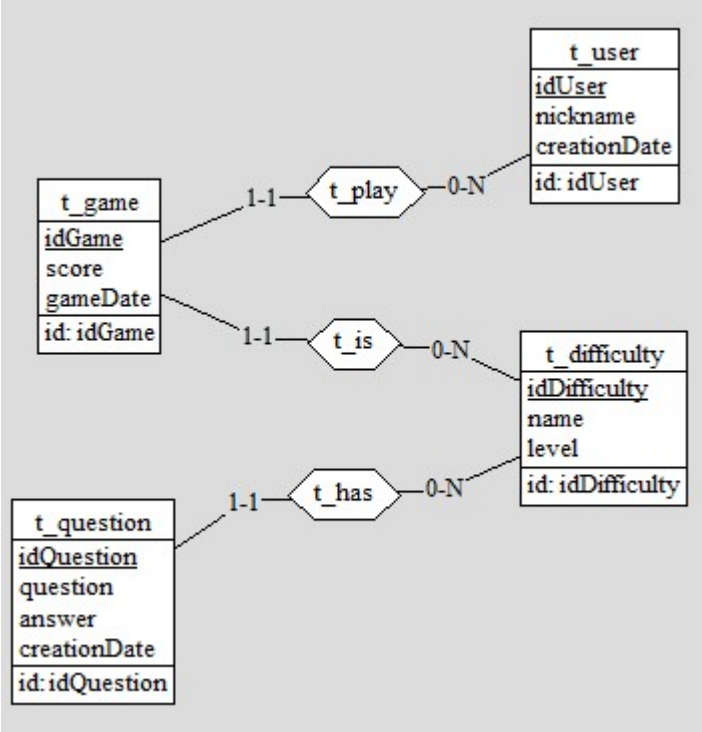

Figure 5 : MCD de la DB

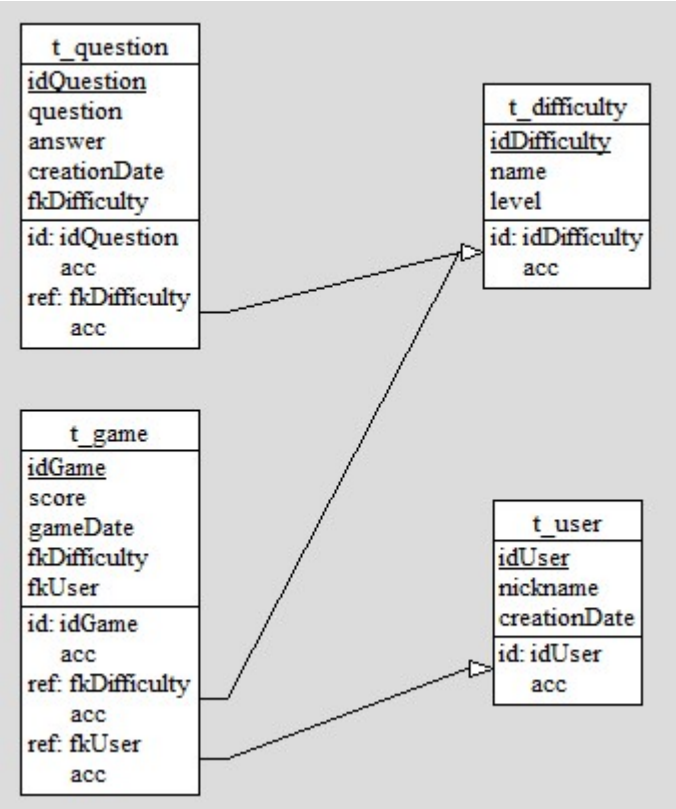

Figure 6 : MLD de la DB

La table t_user contient les colonnes suivantes :

Nom	Type	Longueur	Description
idUser	Int (Auto increment)	11	Id de l'utilisateur
nickname	Varchar	50	Pseudo du joueur
creationDate	Date	-	Date de création du profil

La table t_difficulty contient les colonnes suivantes :

Nom	Type	Longueur	Description
idDifficulty	Int (Auto increment)	11	Id de la difficulté
name	Varchar	25	Nom de la difficulté
level	int	10	Niveau de difficulté

La table t_question contient les colonnes suivantes :

Nom	Type	Longueur	Description
idQuestion	Int (Auto increment)	11	Id de la question
question	Varchar	250	Énoncé de la question
answer	Varchar	250	Réponse à la question

CreationDate	Date	-	Date de création de la question
fkDifficulty	int	11	Clé étrangère de la difficulté

La table t_game contient les colonnes suivantes :

Nom	Type	Longueur	Description
idGame	Int (Auto increment)	11	Id de la partie
score	int	10	Score du joueur pour la partie
gameDate	Date	-	Date de la partie
fkUser	Int	11	Clé étrangère du joueur
fkDifficulty	int	11	Clé étrangère de la difficulté

Une fois la DB créée, des scripts seront exportés afin de pouvoir refaire la DB sur un autre serveur.

Étant donné qu'il y a une base de données, il faudra inclure un fichier de log et la gestion d'erreurs dans le programme. En cas de problème avec la DB, le programme crashera. C'est pourquoi l'utilisation d'un fichier de logs est nécessaire pour résoudre ces erreurs ou empêcher le programme de crash lors de son utilisation. Il faudra donc une gestion de documents extérieur au projet (ExternalManager) pour écrire dans le fichier.
La classe ExternalManager permettra également de récupérer les informations du fichier config.ini.

Lors de la mise en place de la DB dans le code, il faudra utiliser la classe MySqlConnection (qui vient du using MySql.Data.MySqlClient) afin de pouvoir configurer et effectuer une connexion à la DB. Une fois cela mis en place, on pourra ouvrir la DB afin d'y effectuer les actions souhaitées.

Une fois la base de données est en place dans le code, une fonction sera créée par commande SQL. Voici une liste des commandes que l'on peut déjà prévoir (les nomspeuvent changer):

Nom	Description
StockPlayer	Stock le joueur
StockGame	Stock la partie
FindPlayerIDWithName	Trouve l'ID du joueur avec son pseudo/nom
FindDifficultyIDWithLevel	Trouve l'id d'une difficulté avec son niveau
GetHighscore	Récupère les highscores (se fera selon une difficulté donnée)
GetAllQuestions	Récupère les questions (se fera selon une difficulté donnée)

4.3.2.4 Autre ?

Voici une liste des classes que l'on peut prévoir :

- Program (par défaut)

- GameManager
- MenuManager
- DatabaseManager
- ExternalManager
- Une classe parent pour les tetriminos et une classe enfant par tetriminos

Cette liste n'est pas exhaustive et des classes vont sûrement se rajouter lors de la réalisation. Néanmoins, celles-ci seront très probablement présentes.

Afin d'éviter des problèmes visuels dû à la console, on peut également prévoir que la console aura une taille de 1920 pixels par 1080 pixels (résolution standard d'écran d'ordinateur) et qu'elle sera placée au centre de l'écran (donc la console va prendre tout l'écran). On ne pourra pas « maximiser » ou changer la taille de la fenêtre. Bien que cela pourrait poser des problèmes sur des écrans avec une résolution plus haute, le jeu étant prévu pour des élèves d'informatique avec des écrans standards, la résolution 1920 par 1080 a été choisie.

Un diagramme d'activité a également été fait et est disponible en annexe.

- Ce paragraphe décrit le fonctionnement de manière détaillée.
  - Autant que possible de manière graphique, imagée, tableaux, etc.
  - Tous les cas particuliers devraient y être spécifiés…
- Il s'agit d'y présenter les fonctionnalités à développer :
  - Découpage en étapes, en modules, en fonctionnalités, etc.
  - Schémas de navigation, schémas événementiels, structogramme, pseudocode, etc.
- Si le projet inclut une base de données :
  - Dictionnaire des données
  - Modèle conceptuel des données, modèles logique des données.

4.4 Conception des tests

Tableau des tests à réaliser lors du projet. Les résultats seront inscrits dans la partie test de ce document.

Nom du Test	Fonctionnalité testée	Description	Condition de réussite	Importance (0 = bas, 5 = haut)
Menu	Menu	Le joueur peut se déplacer dans le menu ainsi que quitter le jeu via le menu principal	Le joueur peut accéder à toutes les parties du menu et des sous-menus	4
Options	Menu d'options	Différentes options peuvent être sélectionnées. Celles-ci impacteront le jeu	Les options en jeu correspondent bien à celles choisies par l'utilisateur	3
Pseudo	Pseudo du joueur	Le joueur possède un pseudo unique. Il peut le définir lors du lancement du jeu	Le pseudo défini par le joueur est unique. Lors du	3

			premier lancement du jeu, il le défini	
Déplacement des pièces	Déplacement et pivotement des pièces	Le joueur doit pouvoir déplacer les pièces horizontalement, les faire pivoter, et accélérer leur descente	Le joueur peut utiliser les touches choisies dans le menu « options » pour effectuer les différentes actions	5
Questions	Questions	Des questions apparaissent lors de la complétion d'une ligne	Les questions s'affichent et le joueur peux y répondre. Elles proviennent de la DB	4
Score	Changement du score	Incrémentation du score lors d'une bonne réponse à une question	Si le joueur répond correctement à une question, les score augmente en fonction de la difficulté	2
Game over	Activation au bon moment	Lorsqu'une pièce a atteint le haut de la zone, la partie est terminée	Le game over s'active uniquement lorsqu'une pièce touche le haut de la zone. La partie est ensuite sauvegardée en DB	3
DB	Bon fonctionnement de la DB avec le jeu	La connexion avec la DB est faite correctement. Des informations doivent pouvoir être récupérées et insérées	Les informations sont correctement récupérées et insérées. Un fichier de log est prévu en cas de problème	4

Les tests seront faits au fil du temps et dans un ordre logique (certains tests ne pouvant être fait avant d'autres dû à un ordre d'implémentation des fonctionnalités). Les tests peuvent être réalisés soit par une personne interne au projet (développeur, CDP, expert), soit par une personne externe (camarade, famille, profs, ect…), soit via un test unitaire. Le personne/outils ayant réalisé le test sera mentionné dans le tableau de résultat.

5 RÉALISATION

5.1 Mise en place du Github

Afin d'assurer une sauvegarde du projet, un repository GitHub a été fait dès le début du projet. Celui-ci a été mis en Public afin que les experts et la CDP puissent y avoir accès sans pour autant avoir besoin d'être ajoutés au projet. Le Git contient non seulement le projet Visual Studio mais également la documentation. Un ReadMe avec une très brève description du projet est disponible sur le repository.

Dû à un problème avec la branche main, le projet se trouve sur la branche master. Le problème était qu'il n'est pas possible d'accéder à la branche main via Visual Studio ou via GitBash. Après des tentatives infructueuses pour copier la branche master sur main, la branche master est devenu la branche sur laquelle le projet a été réalisé. Cette décision a été prise pour plusieurs raisons :

- Il n'y a qu'une seule personne sur le projet. Cela n'implique donc pas de problèmes possibles liés à des pushs simultanés ou au fait que plusieurs personnes travailleraient sur la même branche.

- Le coût temps-efficacité-risques. En effet, le github devant être fait le premier jour selon la planification, le temps passé pour régler ce problème ne serait pas optimal en termes d'efficacité (principe de Pareto). Les potentiels risques causés par l'utilisation de la branche master sont également minimes (voir nuls, mais le risque 0 n'existe pas).

En outre, si le projet avait été un projet de groupe, le temps passé à régler le dit soucis aurait été rentable pour l'efficacité et la réduction de risques par la suite, ce qui n'est pas le cas ici.

5.2 Mise en place de Visual Studio 2022

Pour la création du projet Visual Studio, le template choisis est le console.Framework et non le console.Net.

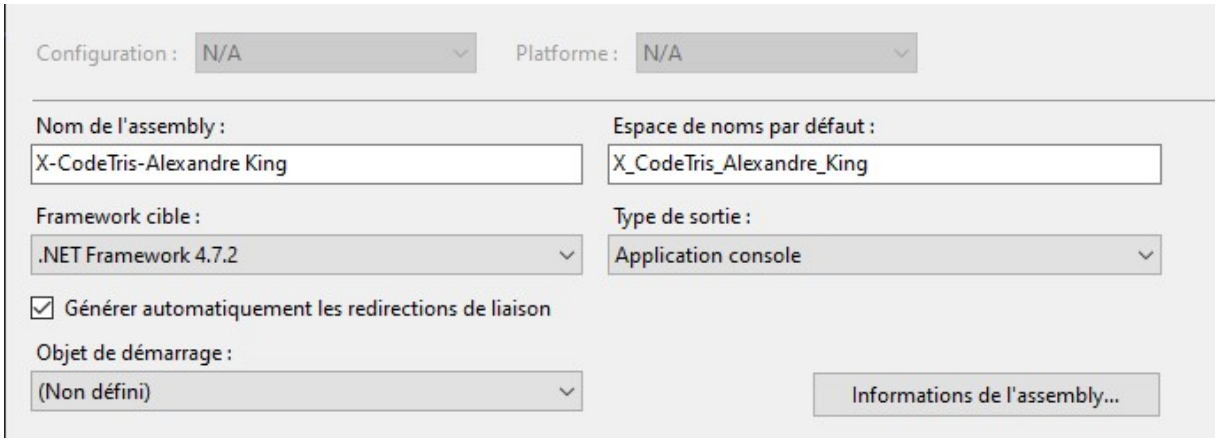

Figure 7 : Configuration dans Visual Studio

Cette décision a été faite en prévision de l'ajout de musique dans l'application. Ayant eu des problèmes pour ajouter du son avec le .Net dans le passé, le .Framework était une certitude pour l'ajout de son de la manière la plus simple et fluide possible.

Une fois le projet crée, il a été lié au Github via l'interface de Visual Studio afin de pouvoir utiliser l'outil Git.

Figure 8 : Onglet Git dans Visual Studio

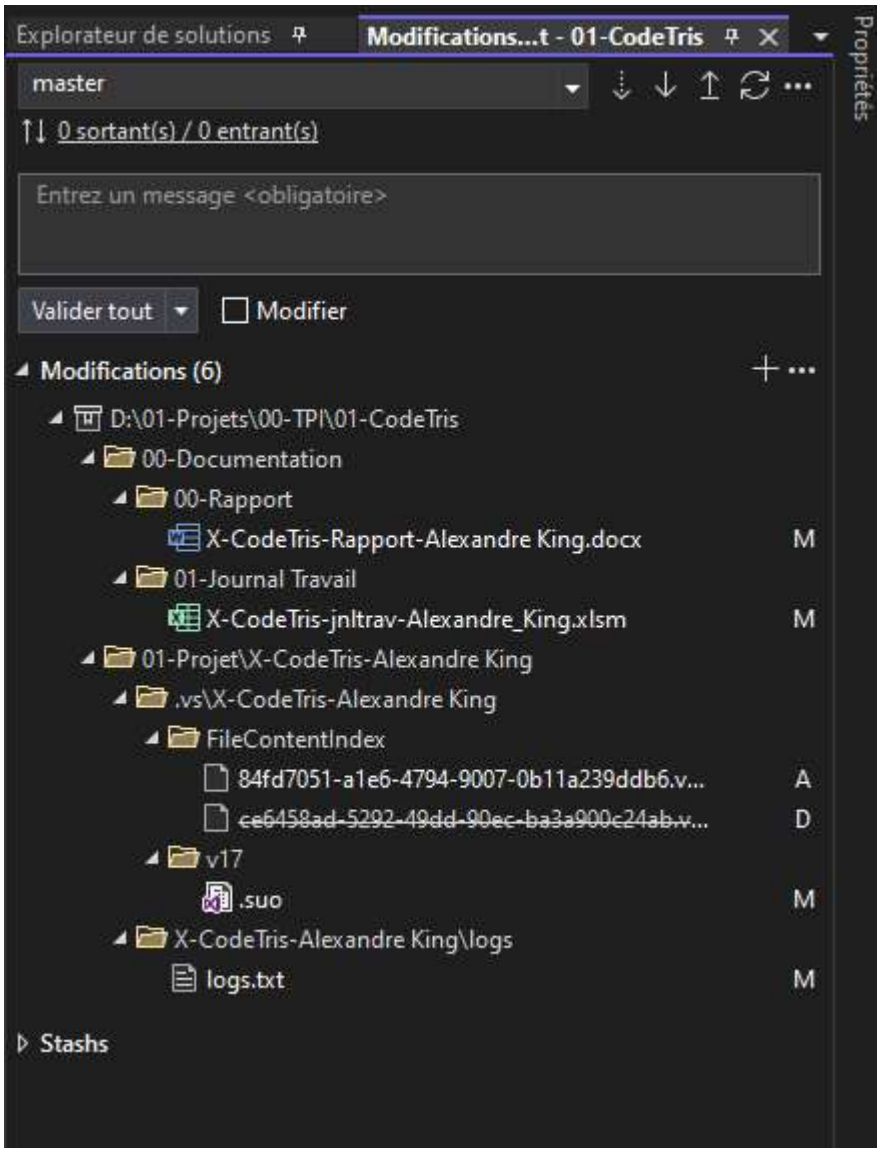

Figure 9 : Outil Git dans Visual Studio

5.3 Mise en place de la base de données (MPD)

Avec l'aide du MLD (point 4.3.2.3), la base de données a été faite dans PHPMyAdmin. Un serveur UWamp a permis d'héberger la DB.
Pour des raisons de sécurité, un utilisateur admin a été créé afin de pouvoir accéder à la DB (évite l'utilisation de l'utilisateur root, mot de passe root).
-Nom : Admin123
-Mot de passe : Admin !123

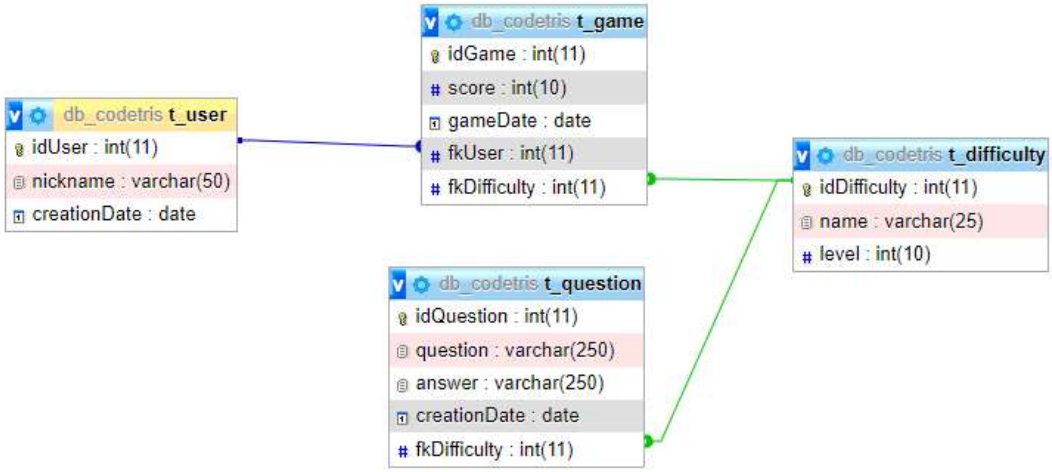

Figure 10 : MPD de la DB

Une fois les tables créées, les clés étrangères ont été ajoutées.
Commande SQL pour l'ajout de clé étrangère :

ALTER TABLE « table visée » ADD FOREIGN KEY « champ de la table» REFERENCES « table étrangère(champ de référence) » ;

Exemple avec l'ajout de la FK « fkUser » sur la table t_game :

ALTER TABLE t_game ADD FOREIGN KEY fkUser REFERENCES t_user(idUser) ;

Les difficultés ainsi que quelques questions ont directement été ajoutées. Les difficultés sont les suivantes :
- Facile (niveau 1)
- Moyen (niveau 2)
- Difficile (niveau 3)

Chaque question à un niveau de difficulté attribué (fkDifficulty). Les questions ont été pensés pour des élèves de première année en informatique. Il existe 3 questions par difficulté (donc 9 au total).

5.4 Premières étapes

Cette partie expliquera les premières étapes après la création du projet Visual Studio, qu'est qui a été mis en place et pourquoi ?

Comme dit dans la conception plusieurs managers permettront de gérer différents points du projet. Voici ceux-ci qui ont été créé dès le départ :
- MenuManager
- GameManager
- DatabaseManager
- ExternalManager

Une partie du code fait lors d'un projet de pré TPI a été repris pour les managers (sauf pour le GameManager car le jeu n'est pas le même). Le code sera expliqué dans la section liée à la fonctionnalité en question.

Dans la classe program.cs (classe par défaut), les lignes suivantes ont été ajoutées dès le départ :

```
static void Main(string[] args)
{
    //make the cursor invisible
    Console.CursorVisible = false;

    //name of the window
    Console.Title = "CodeTris";
```

Figure 11 : Premières lignes du program.cs

Ces lignes permettent de cacher le curseur du clavier et de donner un meilleur nom à la console.

Les managers MenuManager et GameManager sont également instanciés au début. Cela permet de n'avoir qu'une seule instance de ces deux classes et d'éviter certains bugs qui pourraient être liés à une multitude d'instances.

Comme évoquer dans la conception, la possibilité de changer la taille de la fenêtre ou de la maximiser doit être retiré afin de limiter les risques de bugs visuels la taille de la fenêtre change. Pour cela, une classe provenant de stackoverflow a été utilisée. Celle-ci accède à des dll du pc afin de retirer ces options sur la console. Une classe stackoverflow a également été utilisée afin de positionner correctement la console sur l'écran après son redimensionnement à la taille maximum de celui-ci. Cette classe récupère le process du programme et change sa position sur l'écran.

5.5 Organisation dans le code

Pour s'organiser au mieux dans le code, différents dossiers permettent de séparer les classes. Par exemple, un dossier contenant tous les managers permet de ne pas avoir à les chercher dans l'explorateur de solution. Cela améliore l'efficacité surtout quand il y a un grand nombre de classes.

Pour certaines fonctionnalités complexes, plutôt que de directement faire le code, une approche avec du pseudo-code et des notes (soit dans Visual Studio soit sur papier) permet de mieux visualiser les étapes de la méthode.

En voici un exemple :

```
0 références
private bool CheckCanMoveInPlayZone()
{
    bool spaceIsEmpty = false;

    //TO DO: Check if next movement will result in the current form (1) to collided with an other form (2 or 3)
    //if no, space is empty and the piece can move
    //else, the piece can move in the wanted direction
    //Check the same for the rotation (Move the piece in the array BUT NOT visually, check, return result?)


    return spaceIsEmpty;
}
```

Figure 12 : Exemple de pseudo code fait avant de commencer certaines fonctionnalités

5.6 Gestion de la base de données dans le programme

La classe DatabaseManager s'occupe de la DB. Elle permet d'effectuer une connexion à la DB afin d'exécuter ensuite des requêtes SQL. Pour effectuer une connexion à la DB, le manager à besoin d'informations sur celle-ci (le server, le nom de la base de données, le nom et le mot de passe d'un utilisateur (avec les droits pour écrire et lire)). Ces données sont récupérées d'un fichier config.ini via le ExternalManager (point 5.7) puis transmises au DatabaseManager. Pour effectuer la connexion, il faut utiliser la classe MySqlConnection provenant du using MySql.Data.MySqlClient.

Lors de l'ouverture de la DB ou d'une requête SQL, une erreur peut se produire (surtout lors de l'ouverture car si le serveur n'est pas en route, la connexion est impossible). Cette erreur fera crash l'application. Pour éviter cela, il faut utiliser un try catch. Il permet « d'essayer » d'exécuter du code et en cas d'erreur, le catch récupère l'exception et exécute un autre code. Dans notre cas, le catch va demander au ExternalManager de noter dans un fichier de logs l'erreur. Ensuite, le programme fonctionnera sans crasher. Cependant, la DB n'étant pas accessible, le jeu se transforme alors en un tetris classique (sans les questions, les tableaux de score et l'enregistrement de parties).

```
static public void OpenDB()
{
    //récup infos depuis config.ini
    if (!_hasConfiguration)
    {
        ExternalManager.LogError("config.ini file isn't correctly configured. Configuration infos hasn't been found !");
        _dbState = false;
    }
    else
    {

        string connectionString;
        connectionString = "SERVER=" + _dbConfigurationInfos["server"] + ";" + "DATABASE=" + _dbConfigurationInfos["database"] + ";" + "UID=" + _dbConfigurationInf
        _connection = new MySqlConnection(connectionString);

        //pas possible de réduire le timeout (possible sur les commandes mais pas la connection)
        try
        {
            _connection.Open();
            ExternalManager.LogInfo("Database opened successfully");
            _dbState = true;
        }
        catch (Exception e)
        {
            ExternalManager.LogError(e.Message);
            _dbState = false;

        }

    }
}
```

Figure 13 : Code de connexion à la DB

Chaque interaction avec la base de données et notées dans le fichier de logs via l'ExternalManager, cela inclut donc :

- Une erreur avec la DB (mauvaise connexion, DB pas trouvé, etc…)
- Une bonne ouverture de la DB
- Une insertion d'informations (nouveau user, une partie, etc…)
- Une sélection pertinente d'informations (comme les questions)

La classe DatabaseManager est une classe statique (static). Consulter le point 5.21.1 pour plus d'informations sur les classes statiques du programme.

5.7 Gestion des éléments externes

La classe ExternalManager permet d'accéder à des éléments extérieurs au programme comme :

- Les fichiers txt (sauvegarde des options, paramètres, etc…)
- Les fichiers de configuration (config.ini)
- Les fichiers de logs (documents txt mais pourrait être remplacés par un autre type)

Ce manager va utiliser le using System.IO afin de pouvoir accéder à des fichiers.
Il va également responsable de la récupération du config.ini. Il va récupérer le fichier brut et le DatabaseManager va ensuite traiter les informations.
Il s'occupe également de noter dans un fichier txt les options choisis par le joueur afin que lorsque celui-ci relance l'application, ces choix soient conservés. Lorsque le joueur modifie une option, le txt est directement mis à jour afin de ne pas perdre l'information.

Il contient également une méthode pour écrire une erreur dans le fichier de logs ainsi qu'une méthode pour écrire une information dans le fichier de logs. Les deux méthodes sont séparées afin que cela soit clair lorsqu'on appelle le manager. (Il aurait été possible d'ajouter un paramètre à la méthode pour désigner s'il s'agit d'une info ou d'une erreur mais il est plus instinctif d'appeler une méthode par ce qu'elle fait. Cela permet aussi d'avoir par la suite un fichier de log pour les erreurs et un pour les infos).

```
/// <summary>
/// Log an error
/// </summary>
/// <param name="error">Error to log</param>
10 références
static public void LogError(string error)
{
    if (File.Exists(LOGS_DIR_PATH + LOGS_FILE_PATH))
    {
        string message = DateTime.Now.ToString() + "\t\t" + "ERROR" + "\t\t" + error;
        using (StreamWriter w = File.AppendText(LOGS_DIR_PATH + LOGS_FILE_PATH))
        {
            w.WriteLine(message);
        }
    }
}
/// <summary>
/// Log an info
/// An info can be something like good insertion in DB or connection to DB successfull
/// </summary>
/// <param name="info">info to log</param>
5 références
static public void LogInfo(string info)
{
    if (File.Exists(LOGS_DIR_PATH + LOGS_FILE_PATH))
    {
        string message = DateTime.Now.ToString() + "\t\t" + "INFO" + "\t\t" + info;
        using (StreamWriter w = File.AppendText(LOGS_DIR_PATH + LOGS_FILE_PATH))
        {
            w.WriteLine(message);
        }
    }
}
```

Figure 14 : Méthodes d'écriture des logs

La classe ExternalManager est une classe statique (static). Consulter le point 5.21.1 pour plus d'informations sur les classes statiques du programme.

5.8 Fonctionnement des menus

Les menus sont gérés par le MenuManager. Un menu est défini dans un dictionnaire<string,int> afin d'avoir une valeur unique par choix. Le manager n'est pas une classe statique, ce qui veut dire qu'il est instancié. Cette instance ce fait dans le program.cs. Lors de son instanciation (donc lorsqu'on appelle le constructeur), une méthode pour chaque menu est appelée afin de définir ses choix. Dans ces méthodes, on ajoute au dictionnaire correspondant un string (nom du choix) ainsi que sa valeur unique en int. Une boucle for passe ensuite dans le dictionnaire et écrit les différents choix sur l'écran.

```
/// <summary>
/// Add all the choices of the option menu
/// </summary>
1 référence
private void DefineOptionMenu()
{
    _optionMenu.Add(0, "Musique");
    _optionMenu.Add(1, "Touches");
    _optionMenu.Add(2, "Difficulté");
    _optionMenu.Add(3, "Retour <=");
}
```

Figure 15 : Ajout de choix possibles pour le menu des options

Lors du lancement de l'application, une boucle do while permet de se déplacer dans le menu. Pour se faire, le curseur est placé à la position du premier choix (la position des choix est stocké dans un tableau dans le manager) et est légèrement décalé à gauche afin de ne pas empiéter sur le choix. Ensuite, une Console.ReadKey récupère l'input utilisateur et permet de savoir si celui-ci souhaite se déplacer vers le bas ou le haut. Si l'input est valide, on supprime le « > », on déplace le curseur et on redessine le signe. Si l'on a atteint une extrémité, on déplace le curseur à l'autre extrémité (donc si l'on est au plus bas, le curseur remonte au haut de la liste de choix). Pour vérifier ça, une variable menuSelector (int) nous indique en permanence où l'utilisateur se trouve dans la liste. On peut donc calculer avec chaque déplacement si l'on sort de la liste de choix ou non et incrémenter ou décrémenter menuSelector en fonction de l'input. Lorsque que le joueur appuie sur la touche ENTER, un switch sur le menu actuel est fait puis, une méthode est appelée afin de gérer le résultat en fonction du menu. On passe la valeur de menuSelector puis on effectue un switch et si l'on appelle un autre menu, celui actuel est effacé et les variables sont redéfini (menuSelector passe à 0 et le nombre maximum d'option est défini par un count du dictionnaire pour savoir le nombre de possibilités). Le nouveau menu est alors affiché et la boucle recommence. Si l'input du joueur n'est pas valide, on efface son input et on replace le curseur (cette partie est faite dans le default du switch).

```
//read the given key
ConsoleKeyInfo keyInfo = Console.ReadKey();
//Depending on which key the user pressed, do something different
switch (keyInfo.Key)
{
    //go to the next option
    case ConsoleKey.DownArrow:
        if (menuSelector < lastIndexMenu)
        {
            menuSelector++;
            //GetMenuOptionPos return a array with the position of the choices
            cursorLeftPos = menuManager.GetMenuOptionPos()[menuSelector];
            Console.SetCursorPosition(cursorLeftPos, Console.CursorTop + 1);
        }
        else
        {
            menuSelector = 0;
            cursorLeftPos = menuManager.GetMenuOptionPos()[menuSelector];
            Console.SetCursorPosition(cursorLeftPos, menuTop);
        }
        Console.Write(">");
        Console.SetCursorPosition(cursorLeftPos, Console.CursorTop);
        break;
        //Go to the previous option
    case ConsoleKey.UpArrow:
        if (menuSelector > 0)
        {
            menuSelector--;
            cursorLeftPos = menuManager.GetMenuOptionPos()[menuSelector];
            Console.SetCursorPosition(cursorLeftPos, Console.CursorTop - 1);
        }
        else
        {
            menuSelector = lastIndexMenu;
            cursorLeftPos = menuManager.GetMenuOptionPos()[menuSelector];
            Console.SetCursorPosition(cursorLeftPos, menuTop + menuSelector);
        }

        Console.Write(">");
        Console.SetCursorPosition(cursorLeftPos, Console.CursorTop);
        break;
```

Figure 16 : Code pour déplacer le curseur dans le menu

```
//ENTER
//Depending on the selected option of the current menu, will do something different
case ConsoleKey.Enter:
    switch (menuManager.GetCurrentMenu())
    {
        case "main":
            ManageMainMenuEnter();
            break;

        case "option":
            ManageOptionMenuEnter();
            break;

        case "highscore":
            ManageHighscoreMenuEnter();
            break;
        case "detailHighscore":
            ManageDetailsHighscoreMenuEnter();
            break;
        case "howToPlay":
            ManageHowToPlayMenuEnter();
            break;

    }
    break;

default:
    //by default, will place the cursor back to the correct position
    Console.SetCursorPosition(Console.CursorLeft - 1, Console.CursorTop);
    Console.Write(">");
    Console.SetCursorPosition(Console.CursorLeft - 1, Console.CursorTop);

    break;
```

Figure 17 : Code pour la gestion du ENTER et en cas d'input invalide

Pour le menu des highscores et le menu de « tutoriel », le processus est légèrement différent. Au vu du fait qu'il faut du texte qui n'est pas une option, le dictionnaire de ces menus contient uniquement l'option pour revenir en arrière et le texte est ajouté par la suite.

```
/// <summary>
/// Shows the how to play menu
/// </summary>
1 référence
private void ShowHowToPlayMenu()
{
    int count = 0;
    foreach (string howToPlayInstruction in _howToPlayMessage)
    {
        Console.SetCursorPosition(Console.WindowWidth / 2 - howToPlayInstruction.Length / 2, Console.WindowHeight / 4 + count);
        Console.Write(howToPlayInstruction);
        count++;
    }
}
```

Figure 18 : Code pour le menu tutoriel

Pour les highscores, c'est le même menu qui est appelé peu importe la difficulté. Cependant, c'est au moment où les scores sont récupérés que la DB sélectionne uniquement les 10 meilleurs pour la difficulté choisie.

Avant de quitter le jeu, un dernier appel au ExternalManager est fait pour réenregistrer les options afin d'assurer que les données soient les bonnes. Ensuite le programme quitte avec la ligne : Environment.Exit(0);

Pour ajouter des couleurs et aider le joueur à reconnaître directement le jeu au démarrage, un effort sur le visuel du menu principal a été fait. Pour cela, un nouveau manager a été créé, le VisualManager. Celui-ci s'occupe d'écrire le titre en ascii ainsi que de dessiner les tetriminos géants. Il est également utilisé dans le

reste du projet pour changer la couleur du texte au de l'arrière-plan. Le VisualManager est une classe statique. Consuler le point 5.21.1 pour plus d'informations.

5.9 Gestion du jeu

Ce point parlera de comment le jeu a été réalisé en partant de la création de l'interface jusqu'au game over en passant par la gestion des tetriminos.

5.9.1 Création de la zone de jeu

Afin de créer une zone de jeu permettant au joueur d'avoir assez d'espace sans pour autant rendre le jeu trop compliqué, plusieurs dimensions ont été testées. C'est finalement une taille de 64 pixels par 52 pixels qui a été retenu. Sachant qu'un carré de tetriminos (chaque tetriminos est formé par 4 carrés) a une taille de 4 pixels par 2 pixels (un pixel de la console est plus haut que large expliquant une taille inégale en pixel), la taille de la zone en fonction de la taille d'un carré de tetriminos est donc de 16 par 26. Les dimensions sont stockées dans des constantes et peuvent être changées à tout moment sans casser le jeu.

Pour dessiner la zone de jeu, l'arrière-plan est mis en gris clair (grâce à une double boucle for et au VisualManager). L'utilisation de l'arrière-plan pour faire la zone de jeu permet de ne pas avoir à réécrire la zone en permance.

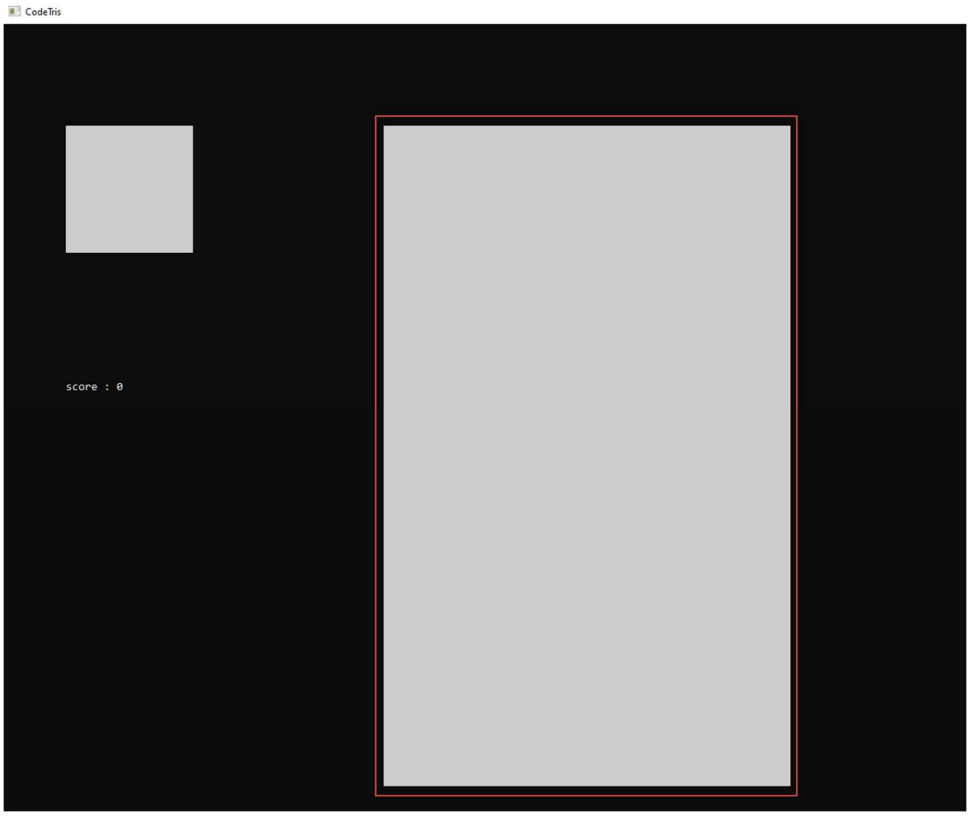

Figure 19 : Zone de Jeu

5.9.2 Création et gestion des tetriminos

Pour dessiner et gérer les tetriminos, un nouveau manager a été créé. Le TetriminosManager permet dessiner et il contient le tetriminos actuel. Cette classe n'est pas statique, elle est instanciée dans le GameManager. Elle a besoin d'être réinstancié au début de chaque nouvelle partie pour repartir sur une base vierge.

Au départ, pour assurer que les tetriminos se dessinait correctement, le O-Block était le seul affiché et utilisé pour les tests de déplacement. Pour le dessiner, le curseur est placé en haut de la zone et à la moitié de la largeur puis, la couleur de texte est changée via le VisualManager pour correspondre à la couleur du tetriminos et finalement, le tetriminos est écrit sous forme de texte avec ce caractère « ▉ » (alt +219).

Dû au fait qu'il y a au total 7 tetriminos, chacun avec les mêmes variables et méthodes, on peut donc faire de l'héritage sur ceux-ci. Une classe parent « Tetriminos » a donc été créée et voici ce qu'elle contient :
Variables :

Name	String
Base Sprite	String[]
Color	String
Current State	Int
Widht	Int
Height	Int
All States	List<string>
Occupation	Bool[,]

Méthodes :

- Constructor()
- ChangeState()
- DefineOccupation

Les variables sont en protected afin de permettre aux enfants d'y accéder et sont déclarés avec des get pour permettre aux autres classes de récupérer la valeur sans pour autant mettre la variable en public. Les méthodes sont en public sauf DefineOccupation() qui est en protected. Elle n'a pas besoin d'être appelé en dehors de la classe. De plus, c'est une méthode virtuelle permettant aux enfants de l'override et de la redéfinir. Cela permet de pouvoir l'appeler dans la classe parent lors du changement d'état de la pièce (ChangeState())

```
/// <summary>
/// Go to the next state (rotate)
/// </summary>
/// <param name="nextState">which state to go to </param>
3 références
public void ChangeState(int nextState)
{
    if (_currentState+nextState >_allStates.Count()-1)
    {
        _currentState = 0;
    }
    else if (_currentState+nextState <0)
    {
        _currentState = _allStates.Count()-1;
    }
    else{
        _currentState += nextState;
    }
    DefineOccupation();
}
/// <summary>
/// Define The occupation of the teriminos in a 4 by 4 square
/// This is in virtual so all child can override it but it can still be call be using the parent class
/// </summary>
9 références
protected virtual void DefineOccupation(){}
```

Figure 20 : Méthode virtuelle du parent Tetriminos

```
override protected void DefineOccupation()
{
    _occupation = new bool[OCCUPATION_SIZE, OCCUPATION_SIZE];
    switch (_currentState)
    {
        case 0:
            _occupation[0, 0] = true;
            _occupation[1, 0] = true;
            _occupation[1, 1] = true;
            _occupation[2, 1] = true;
            _width = 3;
            _height = 2;
            break;
        case 1:
            _occupation[1, 0] = true;
            _occupation[1, 1] = true;
            _occupation[0, 1] = true;
            _occupation[0, 2] = true;
            _width = 2;
            _height = 3;
            break;
        default:
            break;
    }
}
```

Figure 21 : Méthode virtuelle dans les enfants Tetriminos

Chaque tetriminos aura ensuite sa propre classe qui héritera de la classe Tetriminos. Dans les classes enfants, chaque variable est définie avec la bonne valeur. On « dessinera » également tous les états possibles de la pièces (toutes les positions de la pièce quand on la tourne). Dans le cas où l'on aurait besoin d'un espace dans la pièce (comme pour le Z-Block par exemple), on utilisera « !! » dans le string pour indiquer plus tard que l'on a besoin d'un espace ici.

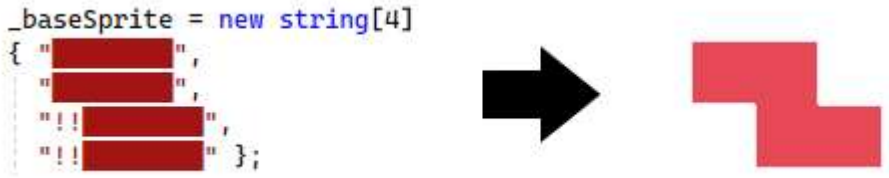

Figure 22 : Exemple d'espace sur les pièces

5.9.3 Déplacement des tetriminos

Pour le déplacement, au moment où le tetriminos doit se déplacer, il est d'abord effacé (comme s'il était dessiné mais avec du vide) puis redessiné à la nouvelle position. Les possibles déplacement sont les suivants :

- Droite
- Gauche
- Bas
- Bas (forcé par le jeu) (appelé bas naturel)
- (Descente directe, succession d'instructions bas. Pas un nouveau mouvement)

Dans un premier temps, lorsque l'utilisateur appuyait sur une touche, le déplacement se faisait à ce moment-là directement (après vérification de si c'était autorisé ou non). Cependant, il arrivait parfois que le bas naturel devait se faire en même temps qu'un autre déplacement (dû au thread de l'input utilisateur (voir point 5.9.5)). Pour résoudre ce problème, il fallait que chaque instruction de déplacement se fasse l'une après l'autre. C'est pourquoi un nouveau thread qui s'occupe uniquement des déplacements des tetriminos a été créer. Une liste d'instructions stocke désormais les instructions puis le thread les exécute à la suite. Cela évite que 2 déplacements se produisent en même temps. Lorsque l'utilisateur appuie sur une touche, l'instruction est ajoutée à la liste. De même pour le bas naturel à chaque boucle principal (voir point 5.10). Cette boucle est expliquée plus en détail au point 5.11.

Pour vérifier si la pièce peut se déplacer, on regarde si sa position X se trouve entre la position X de la zone de jeu et la position X + la largeur de la zone de jeu. Si la pièce peut se déplacer, on ajoute l'instruction à la liste. Pour tous les déplacements, il a également fallu vérifier s'ils pouvaient s'effectuer sans se faire sur un autre tetriminos déjà posé. Pour cela, un tableau bidimensionnel de int représentant le jeu visuellement a été fait dans le backend. Dans chaque case, on peut trouver les valeurs suivantes

- 0 = case vide
- 1 = case remplie par le tetriminos en train de tomber
- 2 = case occupée par un tetriminos posé
- 3 = case bloquée dû à une mauvaise réponse à une question

5.9.4 Rotation des tetriminos

La rotation des pièces est sûrement le plus gros défi de ce projet. Il a fallu dessiner chaque configuration de chaque pièce (position de rotation possible) puis faire de même dans le tableau de boolean _occupation (définir l'occupation de la pièce pour chaque position).

```
//possible States of the tetriminos (!! is equal to space)
string[] _state1 = new string[6]
{
"  ■■  ",
"  ■■  ",
"■■■■■■",
"■■■■■■",
"  ■■  ",
"  ■■  "
};
string[] _state2 = new string[4]
{
"■■■■■■■■■■",
"■■■■■■■■■■",
"!!■■■■!!!!",
"!!■■■■!!!!",
};
string[] _state3 = new string[6]
{
"!!■■■■!!",
"!!■■■■!!",
"■■■■■■■■",
"■■■■■■■■",
"!!■■■■!!",
"!!■■■■!!"
};
```

Figure 23 : Dessins des états de rotation possible pour le T-Block

```
override protected void DefineOccupation()
{
    _occupation = new bool[OCCUPATION_SIZE, OCCUPATION_SIZE];
    switch (_currentState)
    {
        case 0:
            _occupation[1, 0] = true;
            _occupation[0, 1] = true;
            _occupation[1, 1] = true;
            _occupation[2, 1] = true;
            _width = 3;
            _height = 2;
            break;
        case 1:
            _occupation[0, 0] = true;
            _occupation[0, 1] = true;
            _occupation[1, 1] = true;
            _occupation[0, 2] = true;
            _width = 2;
            _height = 3;
            break;
        case 2:
            _occupation[0, 0] = true;
            _occupation[1, 0] = true;
            _occupation[2, 0] = true;
            _occupation[1, 1] = true;
            _width = 3;
            _height = 2;
            break;
        case 3:
            _occupation[1, 0] = true;
            _occupation[0, 1] = true;
            _occupation[1, 1] = true;
            _occupation[1, 2] = true;
            _width = 2;
            _height = 3;
            break;
        default:
            break;
    }
}
```

Figure 24 : Définition de l'occupation selon l'état de rotation du tetriminos

Dans le tableau d'occupation, true veut dire que la pièce occupe cet espace. Sinon, l'espace est libre. Il faut ainsi définir chaque position dans le tableau qui est occupé. On modifie également la hauteur et la largeur.

Ensuite, lorsque l'utilisateur appui sur la touche de rotation, il faut vérifier s'il est possible de tourner le tetriminos. Pour cela, on va simuler la rotation (tourner la pièce en back, envoyer le tableau d'occupation, retourner la pièce dans l'autre sens), vérifier avec le tableau en backend qui représente la grille de jeu et s'il la pièce peut tourner, alors on l'efface visuellement (comme pour le déplacement), on change sa position en la faisant tourner et on la réaffiche. Dans le cas où la pièce peut tourner mais qu'elle est collée à un bord, la pièce est décalée sur le côté avec de ne pas dépasser de la zone de jeu.

(Le point rouge représente la position (x ; y) du tetriminos)

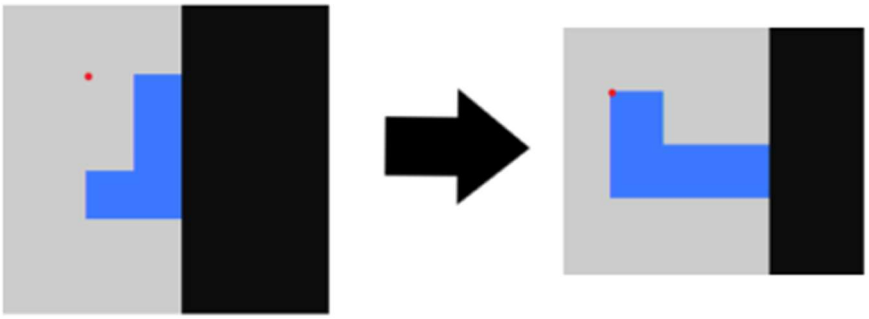

Figure 25 : Exemple de décalage lors de la rotation

Ici, la pièce aurait dû dépasser de la zone lors de la rotation, mais sa position a été décalé de 1 vers la gauche afin d'éviter cela et de quand même permettre au joueur d'effectuer une rotation.

Dans le cadre du O-Block, celui-ci n'ayant qu'une seule position possible, il n'est pas nécessaire de dessiner d'autres états que celui de base. (La pièce ne tourne donc jamais car cela ne changerait rien visuellement)
//Rotation des pièces (dessin de chaque states, modifications du tableau boolean , (parler des !! pour faire un espace/décalage sur certaine pièce)

L'instruction de rotation est également gérée via la liste d'instructions (voir point 5.11).

5.9.5 Inputs utilisateur

Lors de la première gestion des inputs utilisateurs, ceux-ci étaient gérés dans la boucle principale de jeu (c'était également la seule boucle de jeu à ce moment-là du projet). Cependant, cela empêchait les pièces de descendre naturellement à cause du fait qu'il faut récupérer l'input (via un console.ReadKey()). La boucle s'arrêtait donc sur cette ligne à chaque fois. La pièce ne pouvait donc descendre que lorsque l'utilisateur appuyait sur une touche. Et plus de cela, il était impossible de déplacer le tetriminios plusieurs fois avant qu'il ne descende d'un cran. Il était obligé de descendre lorsque qu'il se déplaçait d'une case à gauche ou à droite (empêchant le joueur d'accéder à certains endroits de la zone de jeu).
Pour remédier à ce problème, un premier thread a été mis en place. ManagePlayerInput permet de récupérer à chaque instant une entrée clavier de l'utilisateur et d'agir en conséquence.

La touche est d'abord récupérée avec un Console.ReadKey(), ensuite, elle devrait passée dans un switch. Cependant, pour permettre au joueur de jouer avec les touches qu'il a sélectionné dans les options, des variables pour les touches sont instanciées au début de la partie. En conséquence, il est impossible d'utiliser un switch par la suite car les Cases requièrent des constantes. C'est donc une suite de If Else qui s'occupe de traiter l'input. Au moment de traiter un input, l'on revérifie la position du tetriminos et s'il est possible de le déplacer ou non (dans le cas où la pièce ne peut pas être déplacée dans une direction, l'instruction n'est pas ajoutée à la liste et aide donc à ce que la méthode de gestion des déplacements ne vérifie pas d'instructions inutiles).

```
_pressedKey = Console.ReadKey(true);
//Can't use a switch because the case require to use a constant
//(currently using a variable incase player modify the wanted playing keys)
if (_pressedKey.Key == _moveLeft)
{
    if (_currentTetriminosXPos - _moveXCapacity * 2 > PLAY_ZONE_X_POS)
    {
        _instructionsTetriminos.Add(LEFT_INSTRUCTION);
    }
}
else if (_pressedKey.Key == _moveRight)
{
    if (_currentTetriminosXPos + _moveXCapacity * 2 + TetriminosManager.GetCurrentTetriminosVisualWidth() * 2 < PLAY_ZONE_X_POS + PLAY_ZONE_WIDTH * 2)
    {
        _instructionsTetriminos.Add(RIGHT_INSTRUCTION);
    }
}
else if (_pressedKey.Key == _moveDown)
{
    if (_currentTetriminosYPos + _moveYCapacity < PLAY_ZONE_Y_POS + PLAY_ZONE_HEIGHT - TetriminosManager.GetCurrentTetriminosVisualHeight())
    {
        _instructionsTetriminos.Add(DOWN_INSTRUCTION);
    }

}
```

Figure 26 : Gestion des inputs pour les déplacements

Pour la gestion de la touche espace (qui permet de faire directement descendre le tetriminos tout en bas), c'est une boucle for qui ajoute l'instruction de descente autant de fois qu'il ne reste de lignes à descendre. (Il n'est pas nécessaire de vérifier le mouvement car il est dans tous les cas le dernier pour ce tetriminos. La liste est ensuite vidée une fois les tetriminos posé)

```
else if (_pressedKey.Key == _dropDown)
{
    for (int i = 0; i < PLAY_ZONE_HEIGHT / 2 - _playZoneTetriminosYPos; i++)
    {
        _instructionsTetriminos.Add(DOWN_INSTRUCTION);
    }
}
```

Figure 27 : Gestion de la touche espace

5.10 Boucle principale du jeu

Une boucle principale de jeu permet de gérer la descente naturelle des tetriminos ainsi que leur apparition. Elle est également responsable de la gestion du game

over, du menu pause (revient au menu principal) ainsi que lancer les autres threads (boucles secondaires, voir point 5.11).

Pour rendre le jeu jouable et fluide, il faut « ralentir » le nombre d'actions per seconde. En s'inspirant d'un moteur de jeu comme Unity qui appel une méthode 1 fois par image (dépend donc du nombre d'IPS/FPS (images par seconde/ Frame per second)), on va appeler le boucle principale un certain nombre de fois par seconde. Dans ce but, l'utilisation d'un Thread.sleep() (du using system.Threading) va permettre d'effectuer une « pause » pour un nombre donné de millisecondes. Une variable _frameTiming va nous permettre de définir le nombre de fois par seconde que la boucle est exécutée. On va également pouvoir définir en fonction de la difficulté la vitesse de jeu. Car plus _frameTiming est bas, plus la « pause » est courte et donc plus la vitesse de jeu est rapide. En mode facile, la valeur de _frameTiming est de 250 ms.
À chaque itération de la boucle, l'instruction de bas naturel est ajoutée à la liste d'instructions (après vérification de si l'instruction peut s'exécuter ou non).
La boucle n'est pas censée s'arrêter. Des méthodes devraient être appelées avant et la variable _inGame qui est la variable de la boucle Do While n'est jamais défini à false sauf dans au moment du game over. Cependant, la méthode de game over renvoi une valeur du fait sortir de la méthode du jeu et renvoie au menu principal.

```
private int StartGame()
{
    _inGame = true;
    //count the frames after the tetriminos touched to bottom to let the player 2 frames of action
    int frameCounter = 0;
    while (_inGame)
    {
        //Check if the game is over
        if (_gameOver)
        {
            StopAllThreads();
            PauseGame();
            ShowGameOverScreen();
            return 0;
        }
        //Check if the player wants to go back to the main menu
        if (_returnToMenu)
        {
            StopAllThreads();
            PauseGame();
            ReturnToMainMenu();
            return 0;
        }
        //Check if the game isn't on pause (like when the player is answering a question)
        if (!_isPaused)
        {
            Thread.Sleep(_frameTiming);
            //Does the game has a current tetriminos
            if (TetriminosManager.HasACurrentTetriminos())
            {
                //Can the tetriminos move down
                if (_currentTetriminosYPos < PLAY_ZONE_Y_POS + PLAY_ZONE_HEIGHT - TetriminosManager.GetCurrentTetriminosVisualHeight())
                {
                    _instructionsTetriminos.Add(NATURAL_DOWN_INSTRUCTION);
                }
                //Let a few frame before blocking the tetriminos
                else
                {
                    frameCounter++;
                    if (frameCounter > _frameBeforeNew)
                    {
                        UpdateTetriminosOccupation(true);
                        frameCounter = 0;
                    }
                }
            }
            //Check if can spawn a new tetriminos, if yes, spawn a new one
            if (_canSpawnNew)
            {
                _canSpawnNew = false;
                _instructionsTetriminos.Clear();
                _hasToRedrawGameOverArea = true;
                TetriminosManager.DefineNewTetriminos();
                DrawNewTetriminos();
                DrawNextTetriminos();
                UpdateTetriminosOccupation(false);
                _instructionsTetriminos.Add(NATURAL_DOWN_INSTRUCTION);
            }
            //Check if the threads aren't started and start them
            if (!_threadsStarted)
            {
                StartAllThreads();
                _threadsStarted = true;
            }
        }
    }
    //shouldn't do those line
    Debug.Write("Main While(True) of the game has ended. Attention is required");
    ExternalManager.LogInfo("The main While(True) of the game has ended. This souldn't happend");
    return -1;
```

Figure 28 : Boucle principale du jeu

5.11 Boucles secondaires

Tous les threads sont démarrés au lancement de la partie depuis la boucle principale.

La première boucle secondaire du programme se trouve dans le program.cs. C'est la boucle qui permet aux menus de fonctionner. On ne sort jamais de cette boucle. Lorsque le joueur lance le programme, il rentre dans la boucle, puis se déplace éventuellement dans les menus tout en restant dans cette même boucle. Lorsqu'il lance le jeu, une valeur est retournée à la fin de la partie et il retourne immédiatement dans un menu et donc dans la boucle. Si celle-ci se termine, alors le thread principal du programme se termine et il faut relancer le jeu.

(Moment où le programme rentre dans la boucle de menu)

```
MenuManagement();
//Shouldn't technically do this line in the program
ExternalManager.LogError("Player exit program.cs. Please check. Player shouldn't get out of the do while. A value might not be returned somewhere.");
Console.ReadLine();

///Manage the menus and how the player moves across each menu
void MenuManagement()
{
```

Figure 29 : Extrait du program.cs pour la boucle de menu

Pour les 2 autres boucles secondaires, elles se trouvent dans le GameManager et sont directement liées au jeu. Elles se trouvent dans des threads qui sont lancés au démarrage d'une partie. Elles peuvent être arrêtées lors d'une mise en pause de l'application (comme pour les questions) ou bien d'un retour au menu (soit via la touche escape soit via le game over). Pour des questions d'optimisation, elles sont d'abord arrêtées puis tuées afin de ne pas continuer à occuper de l'espace mémoire du processeur.

La première boucle est celle pour la gestion des inputs utilisateur. Elle est nécessaire pour que le joueur puisse entrer des touches à tout instant de la partie. Les inputs ne doivent pas dépendre d'une autre boucle et il ne doit pas y avoir de délai entre eux.

La deuxième boucle est celle pour la gestion des tetriminos. Celle-ci a un temps de pause de 25 ms afin d'éviter tout risque que deux instructions soient exécutées en même temps (ce qui pourrait causer un bug). À l'intérieur de la boucle, une liste d'instructions exécute à chaque itération l'instruction avec l'id le plus bas (l'id 0 de la liste). Une fois l'instructions faites, elle est supprimée de la liste. Si aucunes instructions n'est présente, rien ne se passe. Cette manière de procédé via une liste d'instructions évite que deux instructions se fassent en même temps (par exemple, un déplacement vers le bas et un déplacement à gauche). Cela causera un premièrement bug visuel mais également un bug fonctionnel car la position du tetriminos est modifiée alors qu'elle ne devrait pas.

```
Thread.Sleep(25);
if (_instructionsTetriminos.Count > 0)
{
    bool isPlaced = false;
    switch (_instructionsTetriminos[0])
    {
        case DOWN_INSTRUCTION:
            if (CheckCanMoveInPlayZone(0, 1, true))
            {
                TetriminosManager.MoveTetriminos(_currentTetriminosXPos, _currentTetriminosYPos, 0, _moveYCapacity);
                _playZoneTetriminosYPos += _moveYCapacity;
                _currentTetriminosYPos += 2;
            }
            break;
        case LEFT_INSTRUCTION:
            if (CheckCanMoveInPlayZone(-1, 0))
            {
                TetriminosManager.MoveTetriminos(_currentTetriminosXPos, _currentTetriminosYPos, -1 * _moveXCapacity, 0);
                _currentTetriminosXPos -= 4;
                _playZoneTetriminosXPos -= _moveXCapacity;
            }
            break;
        case RIGHT_INSTRUCTION:
            if (CheckCanMoveInPlayZone(1, 0))
            {
                TetriminosManager.MoveTetriminos(_currentTetriminosXPos, _currentTetriminosYPos, 1, 0);
                _currentTetriminosXPos += 4;
                _playZoneTetriminosXPos += _moveXCapacity;
            }
            break;
```

Figure 30 : Extrait de la boucle secondaire pour la gestion des tetriminos

On trouver également la méthode pour redessiner la ligne rouge de game over dans cette deuxième boucle.

5.12 Collisions des pièces

//Collisions (grille de 4x4 pour les pièces en boolean, puis comparaison avec la grille de jeu principale)(opti, dire que si on détécte 4 true alors on break (parce que max 4 « pixel » par pièces)
//Aide de CDP sur collisions, bien détaillés la logique derrière
//Correction du bug ou les pièces passent aux travers (agrandissement de la grille en back et définition à 3 (bloquée))

5.13 Complétion des lignes

//Complétion des lignes, 2ème tableau avec les couleurs, réécriture du code pour la gestion de cette partie, vérifier que les lignes se supprime correctement (même si deux lignes sont espacées mais complétées en même temps)

5.14 Blocages des lignes

//Gestion du blocage et de la suppression (ne dois pas supprimer une ligne bloquée)
//optimisation pour le visuel (supprime tout en back puis réécrit 1 seule fois)

5.15 Gestion des questions

Cette partie aborde tous ce qui est lié aux questions. Dans le cas où la base de données de fonctionne pas, le jeu fonctionne mais les questions ne seront pas présentes car elles sont liées à la DB.

5.15.1 Affichage des questions

Afin de gérer au mieux les questions, un QuestionManager a été crée ainsi qu'une classe Question. La classe Question contient un énoncé et la réponse à la question. Le QuestionManager contient une liste de questions, une liste de questions usées et une méthode pour récupérer une question aléatoire.
Lorsque le joueur complète une ligne, le jeu se met en pause et un appel au question manager est fait pour obtenir une question aléatoire. La question est retirée de la liste de question et est ajoutée à la liste des questions utilisées. Si la liste de questions possibles est vide, alors on la reremplie via la liste de questions utilisées et celle-ci est vidée.

```
/// <summary>
/// Get a random question from the list of possibility questions
/// </summary>
/// <returns>a question</returns>
1 référence
public Question GetRandomQuestion()
{
    //First ask for each possible question so that the same question isn't ask more often then an other
    //This also makes the game more fun because it prevents the user from having always the same question
    if (_allQuestions.Count == 0)
    {
        //once each question has been asked, reset the possible question list and clear the used question list
        foreach (Question usedQuestion in _usedQuestions)
        {
            _allQuestions.Add(usedQuestion);
        }
        _usedQuestions.Clear();
    }
    //Once a question has been asked, it is added to the used question list
    int currentQuestionID = random.Next(_allQuestions.Count);
    Question question = _allQuestions[currentQuestionID];
    _usedQuestions.Add(question);
    _allQuestions.RemoveAt(currentQuestionID);
    return question;
}
```

Figure 31 : Gestion de la récupération d'une question

Cela permet d'éviter qu'une même question tombe plusieurs fois de suite en plus d'assurer que toutes les questions seront posées avant de recommencer la liste.

Une fois la question choisie, elle renvoyée au game manager et l'énoncé est affiché. Dans le cas où l'énoncé est très long, le programme l'écrit sur plusieurs lignes.

5.15.2 Mise en pause du jeu

//Mise en pause du jeu (arret des threads, boolean, problème avec les valeurs qui faisait quitter l'application (car fin de la boucle dans program.cs))(optimisation pour pas avoir des threads arreté qui prennent de la mémoire)

5.15.3 Inputs utilisateur (pour sa réponse)

//gestion de la réponse

5.15.4 Vérification de la réponse

Une fois que le joueur a validé sa réponse, celle-ci est envoyer à une autre méthode qui va la comparer à la réponse attendue.
Afin de permettre une marge d'erreur, une méthode utilisant l'algorithme de Levenshtein va calculer le nombre de caractères de différence entre la réponse donnée et celle attendue (trouvée sur stackoverflow). Ensuite, il ne reste plus qu'à définir le pourcentage de marge. Ici, il est de ~12%. Cela permet qu'une réponse avec plus de 12 caractères à le droit à 1 erreur. Pour encore plus aider le joueur, les deux réponses sont mises en caractères minuscules.
Si la différence entre les deux réponses est plus petite que la marge maximum, alors la réponse est considérée comme valide. Sinon, elle est considérée comme fausse.
Si la réponse est valide -> le nombre de bonnes réponses augment, le score augmente et la ligne est supprimée.
Si la réponse est fausse -> le nombre de mauvaises réponses augmente, la réponse attendue est affichée et la ligne la plus basse se bloque.
Pour visuellement indiquer le résultat au joueur, la question s'affichera en vert si la réponse est bonne et en rouge si elle incorrecte.
Pour afficher la réponse attendue en cas de mauvaise réponse, la réponse attendue stockée dans la question s'écrira en dessous de la réponse donnée.
Les méthodes de blocage et de suppression des lignes ne se font qu'une fois toutes les questions répondues (voir point 5.13 pour la complétion et le point 5.14 pour le blocage).

```
/// <summary>
/// Checks the player given answer. The player has a small tolerance on his answer based on the answer lenght
/// (the longer's the correct answer, the more tolerance the player has)
/// </summary>
/// <param name="givenAnswer">The player's answer </param>
/// <param name="correctAnswer">The correct answer</param>
/// <returns></returns>
1 référence
private bool CheckAnswer(string givenAnswer, string correctAnswer)
{
    decimal margin = correctAnswer.Length / 12;
    int tolerance = Convert.ToInt32(Math.Floor(margin) * 2);
    if (LevenshteinDistance.GetDistance(givenAnswer.ToLower(), correctAnswer.ToLower()) <= tolerance)
    {
        //Show green dot
        _numberOfRightAnswer++;
        _score += _pointsToAdd * _completedLineMultiplycator;
        return true;
    }
    else
    {
        //Show Red Dot
        _numberOfWrongAnswer++;
        ShowCorrectAnswer(correctAnswer);
        return false;
    }
}
```

Figure 32 : Méthode de vérification du score

5.15.5 Aparté sur le score

Ce sous-chapitre n'est pas celui qui décrira en profondeur la gestion du score, néanmoins, il reste important d'abordé le score au vu du fait que celui-ci est lié aux questions.
Lorsque le joueur répond correctement à une question, il augmente son score. En fonction de la difficulté, il gagnera plus ou moins de points. Il gagne également un

bonus de multiplication de points selon le nombre de ligne complétées en même temps (donc s'il remplit 1 seule ligne, il gagnera le nombre standard de points, en revanche, s'il remplit 4 lignes en même temps, il gagnera le double de points)

5.16 Game Over

//Game over
//Variables à modifié
//Appel de l'écran de fin (+ profité du visual manager pour le visuel)
//Mise en DB de la partie

5.17 Gestion du score

// augmentation et réécriture du score

5.18 Reset de la partie

Au début de chaque partie, plusieurs variables doivent être remises à leur valeur par défaut. Bien qu'il soit possible de réinstancié le GameManager, il reste plus propre de directement les redéfinir. Voici un exemple de quelques variables qui seront réinstanciées au début de la partie :

- Le score
- Le tetriminos (et sa position)
- Le nombre de bonnes et de mauvaises réponses
- Si le jeu est en marche
- Si le jeu est en pause
- Etc….

Cela permet de recommencer le jeu à l'infini dans le programme.

5.19 Petites optimisations

Afin d'optimiser

//Gestion des timeout de la DB + si la DB n'est pas accessible

5.20 Problèmes rencontrés

//parler des différents problèmes et de comment ils ont été résolus

5.21 Résumé des classes

//résumer des classes finales (tableau avec nom, description, statique/non-statique)

5.21.1 Aparté sur les classes statiques (static)

Qu'est qui a finalement été fait en comparaison avec la conception
Utilisation de classe Static (pourquoi pas singleton ? => perte de temps sur le projet pour des détails mineurs qui apportent peu. Aucuns problèmes liés à une multiple instance de certaines classe). Citer les classes non-static et pourquoi elles ne sont pas static

- Cette partie permet de reproduire ou reprendre le projet par un tiers.
- Pour chaque étape, il faut décrire sa mise en œuvre. Typiquement :
  - Versions des outils logiciels utilisés (OS, applications, pilotes, librairies, etc.)
  - Configurations spéciales des outils (Equipements, PC, machines, outillage, etc.)
  - Code source commenté des éléments logiciels développés.
  - Modèle physique d'une base de données.
  - Arborescences des documents produits.
- Il faut décrire le parcours de réalisation et justifier les choix.

5.22 Points supplémentaires

Cette partie présentera les ajouts supplémentaires fait au jeu ainsi qu'une éventuelle description détaillée de la manière utilisée

Ajout supplémentaire	Date	Description
Musique	02.05.2024	Ajout de musique dans le menu. Gestion de la musique (on/off) dans les options
Prévisualisation du prochain tetriminos	15.05.2024	Désormais, on peut voir quel tetriminos viendra une fois celui en jeu posé
Possibilité de quitter le jeu	22.05.2024	Le joueur peut quitter le jeu pour revenir au menu principal à tout instants

5.22.1 Réalisation de points supplémentaires

5.22.1.1 Musiques et sons

//SoundManager (premier ajout non demandé)
//Gestion des différentes musiques (?)

5.22.1.2 Visualisation de la prochaine pièce

//Améliorations dans le jeu (prochaine pièce, expliquer comment fait)

Afin d'aider le joueur, on peut lui montrer quel tetriminos lui sera donné une fois le tetriminos actuel posé.

Pour cela, dans le TetrminosManager, une deuxième variable de type Tetriminos a été créée (nextTetriminos). Lorsqu'un new tetriminos est demandé, le tetrminos actuel devient celui prévu au préalable et nextTetriminos est redéfini via un random. Pour éviter tout problème au lancement du jeu (dû au fait qu'il n'y pas de tetriminos défini pour la prévisualisation au début) le programme vérifie que nextTetriminos n'est pas null. Concernant l'affichage du tetriminos (visuellement), une zone dédiée a été ajoutée à côté de la zone de jeu.

Figure 33 : Zone de prévisualisation

5.22.1.3 Menu pause (simplifié)

//Touche escape pour quitter (fonctionne à chaque instant, parler de ça (dire qu'un thread aurait pu être fait juste pour la touche mais pas opti))

- Historique des modifications demandées (ou nécessaires) aux spécifications détaillées.
- Date, raison, description, etc.

6 TESTS

6.1 Dossier des tests

6.1.1 Procédure

La procédure des tests est expliqué afin de comprendre la bonne démarche à effectuer.

6.1.1.1 Menu

Procédure de test pour le menu :

1. Tester des bouger haut en bas avec les flèches
2. Si la flèche est tout en bas ou tout en haut, elle revient au prochain choix possible (si tout en haut, revient tout en bas et inversement)
3. La touche enter permet d'accéder au sous-menu
4. Le sous-menu est celui attendu
5. Il est possible de revenir en arrière

6.1.1.2 Options

Procédure de test pour les options:

1. La touche enter permet de changer l'option sélectionnée
2. L'option passe correctement au prochain choix (ON/OFF pour le son (voir couleur), autre possibilité pour les autres)
3. Les options choisies sont les bonnes une fois en jeu. (La musique s'active directement, les touches sont les bonnes. Pour la difficulté, voir vitesse du jeu ou questions (voir DB))

6.1.1.3 Pseudo

Procédure de test pour le pseudo :

1. Un pseudo aléatoire est attribué lors du premier lancement.
2. Il est unique (voir en DB)
3. Il reste le même, tant qu'il n'est pas modifié dans le param.txt

6.1.1.4 Déplacement des pièces

Procédure de test pour le déplacement des pièces :

1. Les pièces peuvent se déplacer grâce aux touches choisies dans les options
2. Elles ne sortent pas de la zone de jeu
3. Elles descendent naturellement toutes seules
4. Elles s'empilent correctement
5. Elles peuvent tourner sur elles-mêmes

6.1.1.5 Questions

Procédure de test pour les questions:

1. Lors de la complétion d'une ligne, une question est posée
2. Elle provient de la DB (voir DB)
3. Elle est du bon niveau de difficulté (voir DB)
4. Le joueur peut y répondre
5. En cas de bonne réponse, elle devient verte et des points sont attribués
6. En cas de mauvaise réponse, elle devient rouge, la bonne réponse s'affiche et une ligne se bloque
7. Une marge d'erreur est accordée au joueur selon la longueur de la réponse attendue (min 12 charactères)

6.1.1.6 Score

Procédure de test pour le score :

1. Lancer une partie (avec et/ou sans DB)
2. Compléter une ligne
3. Si DB active, répondre correctement à la question (voir DB pour réponse)
4. Voir si le score à gauche augmente
   (doit augmenter en fonctionner de la difficulté et du nombre de lignes compléter à la fois)
5. Faire un Game over
6. Voir si le score final s'affiche

6.1.1.7 Game Over

Procédure de test pour le Game Over :

1. Lancer une partie
2. Empiler les pièces jusqu'à la ligne rouge (doit dépasser cette dernière)
3. L'écran de Game Over doit s'afficher

4. (Si des questions ont été posées, les nombre de bonnes et mauvaises réponses doit être juste. Pour vérifier, lancer la partie avec la DB, compléter des lignes, vérifier les réponses à la fin)
5. Le score doit s'afficher
6. Si le joueur appui sur une touche il doit retourner au menu principal

6.1.1.8 DB

Procédure de test pour la DB:

1. Utiliser les scripts de création et de remplissage de la DB
2. Configurer le config.ini
3. Lancer le jeu, si le « DB » en haut à gauche est vert, OK
4. Faire une partie
5. Les questions doivent provenir de la DB (voir DB)
6. Voir les highscores pour la difficulté de la partie qui vient d'être faite
7. La partie doit s'afficher

6.1.2 Résumé des tests

Cette partie résumera les tests effectués et permettra d'effectuer un bilan de ceux-ci.

Nom du test	Date du test	Passation	Commentaire
Menu	13.05.2024	OK	Il est possible de parcourir tout le menu et de de revenir en arrière. Le sous-menu « score » est plus lent si la DB n'est pas accessible mais reste fonctionnel.
Options	13.05.2024	OK	Il est possible de modifier chacune des options. Celle-ci est directement enregistrée dans le fichier de paramètre et prend effet immédiatement (pour le son par exemple).
Pseudo	17.05.2024	OK	(Peut être amélioré) Un pseudo aléatoire est attribué au joueur. Le joueur ne peut pas encore le modifié.
Déplacement des pièces	17.05.2024	OK	Les pièces peuvent être déplacées horizontalement et vers le bas (touche du bas ou espace pour faire descendre directement). Elles ne sortent pas de la zone de jeu. Elles peuvent également tourner (il reste un bug si la pièce est tout en bas, elle sort de la zone

			si on la tourne, le jeu ne crash pas)
Questions	17.05.2024	OK	Les questions sont prises depuis la BD. Elles sont ensuite choisies en fonction de leur niveau de difficulté. Une question aléatoire est posée au joueur.
Score	17.05.2024	OK	Le score augmente si le joueur répond correctement à une question. Plus la question est dure, plus le joueur gagne de points
Game over	17.05.2024	OK	Si une pièce dépasse la ligne rouge, l'écran de fin est affiché avec le score final, le nombre de bonnes et de mauvaises réponses et la possibilité de retourner au menu principal
DB	17.05.2024	OK	Les informations sont bien récupérées depuis la DB. Les parties sont enregistrées en DB. En cas d'erreur avec celle-ci, le jeu ne crash pas et l'erreur est notée dans un fichier de log.

7 CONCLUSION

7.1 Bilan des fonctionnalités demandées

Un bilan final des fonctionnalités permet de voir quelles tâches demandées ont pu être effectuée/complétée ainsi que d'estimer les temps qu'il faudrait pour compléter celle qui ne le serait pas.

Fonctionnalité	Complétion	Temps supplémentaire nécessaire	Commentaires
Déplacements des pièces	95%	10h	Bien qu'il reste des petits bugs, la fonctionnalité est présente. (Principe de pareto pour les 5% restant)
Game Over	100%	-	
Complétion des lignes	100%	-	

Questions	100%	-	Les questions sont prises depuis la DB. Si celle-ci n'est pas active, le jeu fonctionne tout de même
Menus	100%	-	
Score	100%	-	Le score est incrémenté par les questions correctement répondus ou bien la complétion de ligne (sans DB)
Options	100%	-	

7.2 Bilan de la planification

La planification a été plutôt juste. La réalisation se rapproche beaucoup de ce qui a été planifié et il n'y a pas de deltas importants.
Les petits deltas sont les suivants :

- Gestion de la DB. En effet, grâce à un pré TPI, la plupart de la DB été déjà prête (commande SQL, méthode de base, etc…)
- Correction de bugs/Amélioration/Optimisation. Cette tâche permettait de laisser une marge lors du projet. Il projet s'étant mieux passé que prévu, les marges étaient un peu trop larges finalement. Mais il est préférable d'avoir trop de marge que pas assez.
- Test. Les tests ne prennent pas en compte des tests fonctionnels par des personnes extérieurs au projet comme la CDP.

Malgré une absence en début de projet et un traitement intense en lien avec cette absence, il n'y pas eu de répercussions réelles sur le projet.

//Mettre bilan graphique en screen ici ou bien annexes ?

7.3 Analyse post-projet

Une analyse à froid sur le programme permet de revenir dessus et voir le projet avec un œil nouveau.
Si l'on compare la conception et la réalisation, l'idée d'un tableau bidimensionnel en backend paraissait évidente pour la gestion des pièces. Une bonne implémentation a permis un rendu final plus que correct ainsi qu'une certaine optimisation non négligeable. A contrario, l'idée de pouvoir exécuté le jeu de manière séquentiel était une erreur lors de l'analyse. Mais une fois le développement du programme commencé, l'on se rend vite compte du problème et l'on n'est pas pénalisé par la suite.
Pour revenir en bref sur les problèmes « majeurs » du projet, le git n'était effectivement pas implémenter comme il le faudrait mais les répercussions étant moindres, il ne semblait pas nécessaire d'y passer plus de temps. Cette décision n'a pas engendré de complications par la suite. En revanche, il est vrai que dans le cas où le projet aurait été un travail de groupe, il aurait été judicieux d'y consacré plus de temps. Un autre problème notable est la gestion des collisions. En effet, les collisions étant compliqué à gérer de base, la console n'aide pas. C'est

pourquoi il serait intéressant de recréer le jeu avec une technologie plus adaptée (comme un moteur de jeu. Unity, Unreal Engine, etc…) afin de voir la différence.

7.4 Bilan personnel

J'ai beaucoup aimé travailler sur CodeTris. Cette application avait des problématiques intéressantes à résoudre et une grosse partie d'algorithmie. Étant quelqu'un qui aime résoudre des problèmes de code, j'ai apprécié me creuser la tête pour trouver des solutions.
Bien qu'étant arrivé préparer grâce un pré-TPI, j'étais quand même nerveux à l'idée de ne pas réussir à implémenter certaines fonctionnalités. De plus, le projet étant sur une durée de 88h, il est également compliqué de ne pas y penser en dehors des heures prévues et cela empêche parfois de se reposer lorsque l'on est chez soi.
Si ce projet était à refaire, ce serait avec grand plaisir. Les fonctionnalités demandées sont bonnes et réalisables après une formation de 4 ans. Cependant, il vrai que l'utilisation de la console complique beaucoup les choses et qu'il existe sûrement des technologies plus adaptées.
Si le temps me l'avais permis, j'aurais aimé ajouter des fonctionnalités en plus ainsi que corriger les bugs de collisions.
Je remercie Madame Curchod pour son soutient lors de ce projet.

8 DIVERS

8.1 Table des illustrations

8.2 Journal de travail

Le journal de travail est disponible en annexe.

8.3 Webographie

//tableau avec nom du site et lien de la page concernée

GitHub :
https://github.com/Xale2111/CodeTris

Text to ascii :
https://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20

Stackoverflow :
https://stackoverflow.com/questions/2344320/comparing-strings-with-tolerance

https://stackoverflow.com/questions/68265841/c-sharp-override-with-different-child-parameters-same-parent

https://stackoverflow.com/questions/38426338/c-sharp-console-disable-resize

https://stackoverflow.com/questions/67008500/how-to-move-c-sharp-console-application-window-to-the-center-of-the-screen

LucidChart (Schéma UML) :
https://www.lucidchart.com/pages

Figma :
https://www.figma.com/

Convertisseur Youtube to Wav :
https://youtubetowav.com/

Musique Tetris :
https://www.youtube.com/watch?v=7TqGvfx1Xvs

Musique suspense :
https://www.youtube.com/watch?v=xjSRwF0pHtc&list=PLo1hNEB444aUfipwqyi6V5JAdjJ8c72Su&index=4

- Références des sites Internet consultés durant le projet.

9 ANNEXES

//git hub (public), JRNLTRV, DB (script + readMe pour setup),Glossaire, GANTT de la planif (en A3 ?), CDC,

- Listing du code source (partiel ou, plus rarement complet)
- Guide(s) d'utilisation et/ou guide de l'administrateur
- Etat ou « dump » de la configuration des équipements (routeur, switch, robot, etc.).
- Extraits de catalogue, documentation de fabricant, etc.