CodeTris

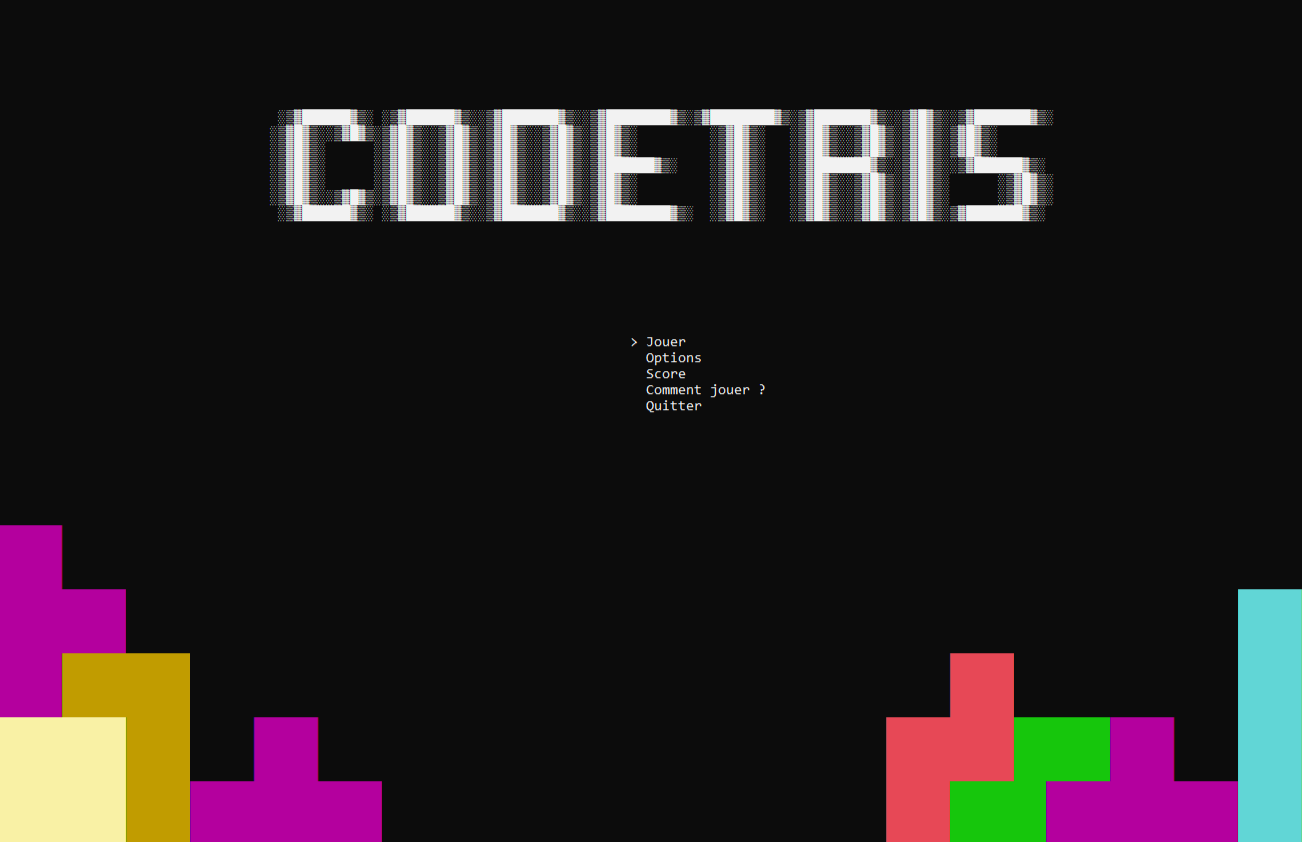


Figure : Image représentant le projet

Alexandre King – MID4

Avenue de Valmont 28b, 1010 Lausanne

88h

Aurélie Curchod

Table des matières

[1 Résumé 3](#_Toc166839933)

[2 Spécifications 4](#_Toc166839934)

[2.1 Titre 4](#_Toc166839935)

[2.2 Description 4](#_Toc166839936)

[2.3 Matériel et logiciels à disposition 4](#_Toc166839937)

[2.4 Cahier des charges 4](#_Toc166839938)

[2.5 Les points suivants seront évalués 4](#_Toc166839939)

[2.6 Validation et conditions de réussite 4](#_Toc166839940)

[3 Planification Initiale 4](#_Toc166839941)

[3.1 Répartition du temps 4](#_Toc166839942)

[3.2 Dates importantes 5](#_Toc166839943)

[3.3 Planification dans Excel 5](#_Toc166839944)

[3.4 Méthode de gestion de projet utilisée 5](#_Toc166839945)

[3.5 Modification de la planification 5](#_Toc166839946)

[4 Analyse 5](#_Toc166839947)

[4.1 Opportunités 5](#_Toc166839948)

[4.2 Objectif 6](#_Toc166839949)

[4.3 Document d’analyse et conception 6](#_Toc166839950)

[4.3.1 Analyse de l’application 6](#_Toc166839951)

[4.3.2 Conception de l’application 8](#_Toc166839952)

[4.4 Conception des tests 9](#_Toc166839953)

[5 Réalisation 10](#_Toc166839954)

[5.1 Dossier de Réalisation 10](#_Toc166839955)

[5.2 Points supplémentaires 11](#_Toc166839956)

[5.2.1 Prévisualisation du prochain tetriminos 11](#_Toc166839957)

[6 Tests 12](#_Toc166839958)

[6.1 Dossier des tests 12](#_Toc166839959)

[6.1.1 Procédure 12](#_Toc166839960)

[6.1.2 Résumé des tests 13](#_Toc166839961)

[7 Conclusion 15](#_Toc166839962)

[7.1 Bilan des fonctionnalités demandées 15](#_Toc166839963)

[7.2 Bilan de la planification 15](#_Toc166839964)

[7.3 Bilan personnel 15](#_Toc166839965)

[8 Divers 15](#_Toc166839966)

[8.1 Table des illustrations 15](#_Toc166839967)

[8.2 Journal de travail 15](#_Toc166839968)

[8.3 Webographie 15](#_Toc166839969)

[9 Annexes 15](#_Toc166839970)

# 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.

//Résultats

Après 88 heures de projet, celui-ci est fini à XXX% (dire ce qu’il resterait à faire ou bien ce qui s’est bien passé, ce qui pourrait être amélioré)

# Spécifications

## Titre

CodeTris : Le bloc de la programmation

## 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.

## Matériel et logiciels à disposition

* PC de l’ETML
* Visual Studio 2022
* Github
* Suite office

## Cahier des charges

Voir annexes

## 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

## 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

# 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.

## 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% |

## 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)

## 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 ?

## 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))

## 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.

# Analyse

## Opportunités

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

## 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.

## Document d’analyse et conception

### 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.

#### 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.

#### 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.

#### 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.

#### 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.

#### 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)

### Conception de l’application

#### Menus

Voici des maquettes de l’interface des menus :



Figure : Maquette du menu principal

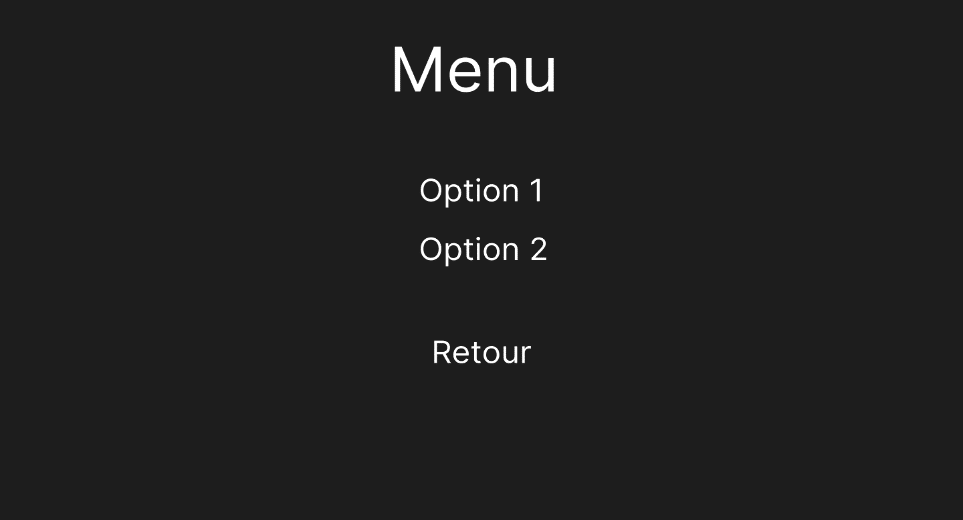


Figure : Maquette des autres menus

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.

#### Jeu

Voici une maquette de l’interface du jeu :

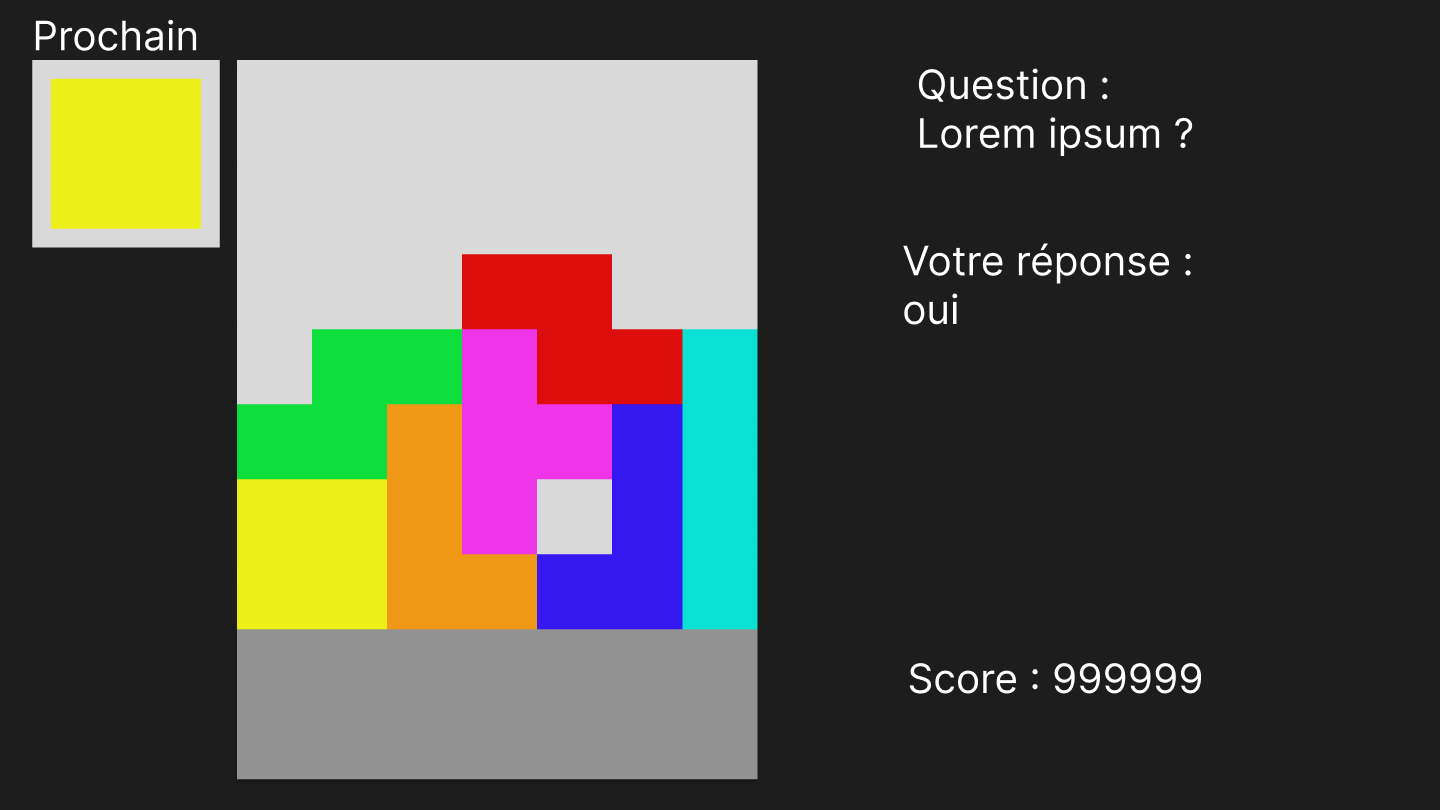


Figure : Maquette du jeu

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.

#### Base de données

Voici le MCD et MLD de la base de données :

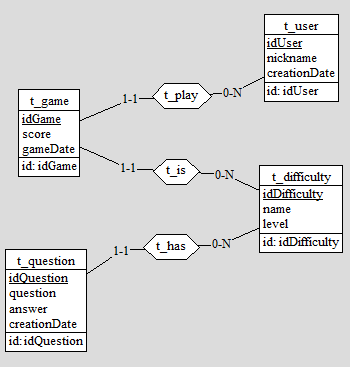


Figure : MCD de la DB

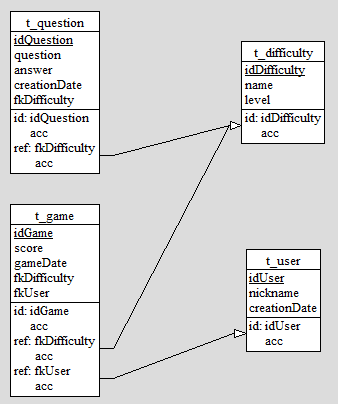


Figure : MLD de la DB

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) |

#### 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.

## Conception des tests

Tableau des tests à réaliser lors du projet. Les résultats seront inscrits dans la [partie test](#_Tests) 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 premier lancement du jeu, il le défini | 3 |
| 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.

## 

# Réalisation

## Dossier de Réalisation

Cette partie abordera le déroulement du projet et la manière dont il a été réalisé.

### Mise en place du Github

//Mise en place du Git (décrire problème (avec screens si possible)

### Mise en place de Visual Studio 2022

//Mise en place du projet VSCode (Framework, liaison avec le git)

### Planif ( ?)

//Parler de la planif

### Maquettes ( ?)

//parler des maquettes (à quoi j’ai pensé en les faisant)

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

//Parler de la DB (MPD, PHPMyAdmin, Uwamp) (Création d’un admin (éviter root, root) Admin123, Admin!123)

//Insert dans la DB

//Création des questions

### Premières étapes

//Program.cs

//MenuManager

//Game Manager

//Classe externe (Stackoverflow)

// En gros, parler du fait que les manager ont été mis en place

### Organisation dans le code

//dossiers

//pseudo code

Image pour montre le pseudo code parfois fait en amont pour aider à la réflexion

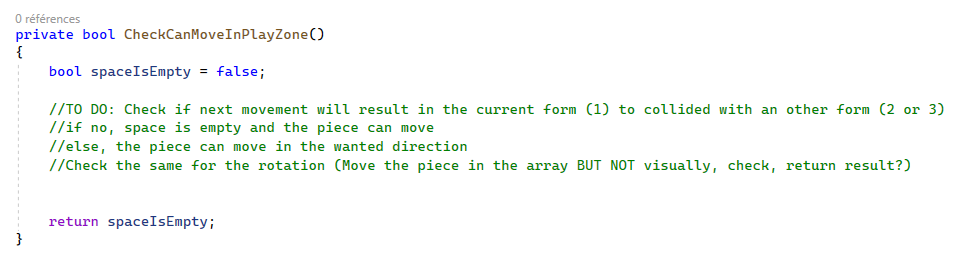


Figure 7 : Exemple de pseudo code fait avant de commencer certaines fonctionnalités

### Gestion de la base de données dans le programme

//External et Database

//Mise en place de la gestion de logs (Error + Info)

//Try Catch DB

### Gestion des éléments externes

//External et Database

//Récup du config.ini via le external

### Fonctionnement des menus

//Menu gestion des premiers trucs (menu principal, déplacements, quitter, revenir en arrière, etc... (processus d’ajout de menu et fonctionnement de ceux-ci))

//Menu des scores (sous-menu avec difficultés, comment c’est gérer etc…)

//Menu principal, Visual Manager

### Gestion du jeu

#### Création de la zone de jeu

//Création de la zone de jeu

#### Création et gestion des tetriminos

//Création des pièces (Tetriminos manager (juste le carré au début)

//Réécriture du code pour la gestion du déplacement des pièces, listes d’instructions, Update

//Héritage sur les tetriminos, classe parent/enfant. Protected, virtual, override

#### Déplacement des tetriminos

//Déplacement des pièces vers le bas

#### Rotation des tetriminos

//Rotation des pièces (dessin de chaque states, modifications du tableau boolean , (parler des !! pour faire un espace/décalage sur certaine pièce)

#### Inputs utilisateur

//Gestion des touches (Ajout de thread finalement sinon le joueur doit attendre que la pièce descende pour bouger) (dire comment pour gérer selon le choix dans les options)(touche espace)

### Boucle principale du jeu

//Thread.Sleep boucle principal avec gestion de frame (inspiré de unity)

//Adaptement de la vitesse du frame timing

### Boucles secondaires

//Boucle du program.cs (dire qu’en soit, on se trouve dans cette boucle à la base et qu’on n’en sort jamais

//Boucle Manage Player Input

//Boucle Manage tetriminios (liste instructions)

### 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))

### 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)

### Blocages des lignes

### Gestion des questions

#### Affichage des questions

//Affichage des question, Questions Manager + classe question

//parler ici des listes des questions usées et des questions possibles

//Parler de la gestion de la taille des questions et réponses

#### 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)

#### Inputs utilisateur (pour sa réponse)

//gestion de la réponse

#### Vérification de la réponse

//Parler de la méthode de permission d’erreurs

//Affichage de la bonne réponse

//Appel de la méthode de suppression si ok (une fois que toutes les questions ont été répondues)

//Appel de la méthode de blocage (comment c’est bloqué)

#### 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.

//Ajout du score, comment c’est gérer en plus en fonction du nombre de ligne

### Suppression et blocage 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)

### 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

### Reset de la partie

//Reset

### Petites optimisations

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

### Problèmes rencontrés

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

### Résumé des classes

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

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.

## 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 |

### Ajouts supplémentaires

#### Musiques et sons

//SoundManager (premier ajout non demandé)

//Gestion des différentes musiques ( ?)

#### Visualisation de la prochaine pièce

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

#### Menu pause (simplifé)

//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))

### Prévisualisation du prochain tetriminos

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.

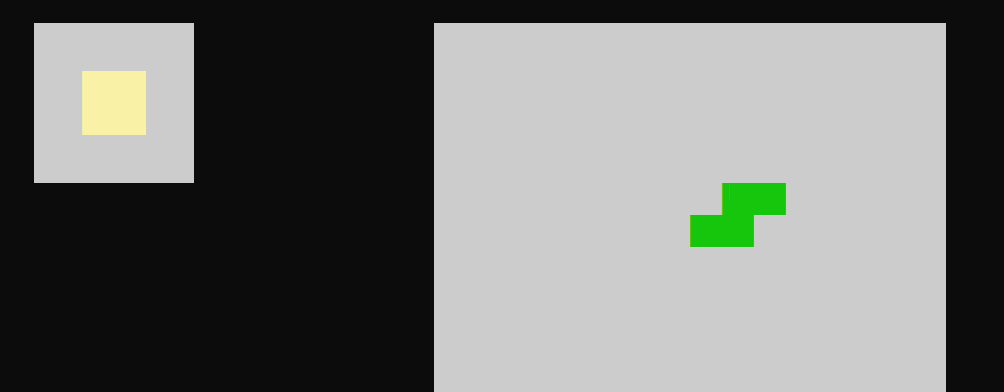


Figure : Zone de prévisualisation

Historique des modifications demandées (ou nécessaires) aux spécifications détaillées.

Date, raison, description, etc.

# Tests

## Dossier des tests

### Procédure

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

#### 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

#### 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))

#### 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

#### 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

#### 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)

#### 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

#### 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

#### 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

### 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. |

On dresse le bilan des tests effectués (qui, quand, avec quelles données…) sous forme de procédure. Lorsque cela est possible, fournir un tableau des tests effectués avec les résultats obtenus et les actions à entreprendre en conséquence (et une estimation de leur durée).

Si des tests prévus dans la stratégie n'ont pas pu être effectués :

raison, décisions, etc.

Liste des bugs répertoriés avec la date de découverte et leur état:

Corrigé, date de correction, corrigé par, etc.

# Conclusion

## Bilan des fonctionnalités demandées

//tableau bilan final (qu’est qui a été fait, qu’est-ce qui reste à faire, pourcentage et temps restant)

Il s’agit de reprendre point par point les fonctionnalités décrites dans les spécifications de départ et de définir si elles sont atteintes ou pas, et pourquoi.

Si ce n’est pas le cas, estimer en « % » ou en « temps supplémentaire » le travail qu’il reste à accomplir pour terminer le tout.

## Bilan de la planification

//comparaison entre planif et réalité, expliquer les deltas importants

Distinguer et expliquer les tâches qui ont généré des retards ou de l'avance dans la gestion du projet. Indiquer les différence entre les planifications initiales et détaillées avec le journal de travail.

## Analyse post-projet

//faire une analyse après la création du programme. Comparée la conception(+analyse) et la réalisation et parlé un peu des problèmes (à nouveau) majeurs

//

## Bilan personnel

Si c’était à refaire:

Qu’est-ce qu’il faudrait garder ? Les plus et les moins ?

Qu’est-ce qu’il faudrait gérer, réaliser ou traiter différemment ?

Qu’est que ce projet m’a appris ?

Suite à donner, améliorations souhaitables, …

Remerciements, signature, etc.

# Divers

## Table des illustrations

[Figure 1 : Image représentant le projet 1](#_Toc166480838)

[Figure 2 : Exemple de pseudo code fait avant de commencer certaines fonctionnalités 11](file:///D:\01-Projets\00-TPI\01-CodeTris\00-Documentation\00-Rapport\X-CodeTris-Rapport-Alexandre%20King.docx#_Toc166480839)

## Journal de travail

Date, activité (description qui permet de reproduire le cheminement du projet), durée, liens et références sur des documents externes. Lorsqu’une activité de recherches a été entreprise, il convient d’énumérer ce qui a été trouvé, avec les références.

## Webographie

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

Références des sites Internet consultés durant le projet.

# Annexes

//git hub (public), JRNLTRV, DB, doc utilisateur (mode d’emploi) ?, 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.