

# Projet informatique STPI2

## Jeu d'Échecs

```
C:\WINDOWS\SYSTEM32\cmd.exe

*-----*
|  | a | b | c | d | e | f | g | h |
| 1 | t | c | f | a | k | f | c | t |
| 2 | p | p | p | p | p | p | p | p |
| 3 |   |   |   |   |   |   |   |   |
| 4 |   |   |   |   |   |   |   |   |
| 5 |   |   |   |   |   |   |   |   |
| 6 |   |   |   |   |   |   |   |   |
| 7 | P | P | P | P | P | P | P | P |
| 8 | T | C | F | Q | K | F | C | T |
*-----*

C'est au joueur blanc de jouer
Selectionnez avec les fleches et la touche espace la piece a bouger
```

# Table des matières

Introduction.....	2
I. Descriptif du cahier des charges.....	3
II. Descriptif de la conception globale.....	3
1) Structure de données.....	4
2) Analyse descendante.....	4
3) Descriptif détaillé des sous programmes.....	5
a) Programme principal.....	5
b) Unité initialisation.....	5
c) Unité jeu.....	6
d) Unité vérification.....	7
e) Unité score.....	7
f) Unité complément.....	7
g) Unité sauvegarde.....	8
h) Unité mise en échec.....	9
III. Guide d'utilisation du programme.....	9
IV. Retour sur le travail de groupe.....	9
Conclusion.....	11

## Introduction

Dans le cadre du projet informatique à effectuer au cours de notre deuxième année à l'INSA, nous avons choisi de réaliser une version numérique du jeu d'Échecs. Lors de la rédaction du cahier des charges, nous nous sommes fixés pour objectif d'être, à terme, en mesure d'encadrer une partie entre deux joueurs.

Par encadrer, nous entendons par là que nous n'avons pas implémenté une intelligence artificielle capable de se substituer à l'un des joueurs, mais que notre programme a pour but de permettre à deux joueurs de jouer aux échecs tout en s'assurant qu'ils respectent les règles.

Dans ce jeu d'échecs, le camp blanc (en minuscule) affronte le camp noir (en majuscule), suivant la réglementation classique des échecs. Le jeu s'arrête lorsque l'un des deux joueurs est mis en échec à la fin de son tour.

Ce rapport présente de manière détaillée nos réflexions, de la conception de l'analyse descendante jusqu'à la réalisation du jeu.

Nous commencerons par un descriptif du cahier des charges initial qui nous permettra de revenir sur les différents changements que nous y avons apporté. Nous analyserons ensuite la conception globale de notre programme. Puis, nous présenterons un rapide guide d'utilisation du programme. Enfin, nous effectuerons un retour sur notre travail d'équipe et notre organisation.

# I. Descriptif du cahier des charges

Le cahier des charges initial prévoyait diverses fonctionnalités telles que « afficher le plateau de jeu et les pièces, déplacer les pièces chacune à leur tour et vérifier la légalité d'un coup, repérer la victoire et enfin écrire le score dans un fichier texte ». En options supplémentaires, nous avons envisagées : « le roque, la promotion, l'accès direct au score d'anciennes parties et l'infographie ». Nous avons respecté notre cahier de charge en réussissant à implémenter l'ensemble des options que nous avons envisagées. En effet, notre jeu d'échec permet bien d'assurer une partie entre deux joueurs en vérifiant que les déplacements effectués soient réglementaires et nous sommes en mesure de détecter une fin de partie. Nous allons justifier les quelques changements apportés au cahier des charges initial.

Les consignes stipulaient l'utilisation d'un fichier texte. Nous avons donc imaginé renvoyer aux joueurs le score (nombre de coups joués en fin de partie et identification du vainqueur) dans un tel fichier. Cependant, nous avons finalement trouvé plus intéressant de créer une nouvelle unité « sauvegarde » grâce à laquelle les joueurs peuvent soit faire le choix d'entamer une nouvelle partie soit celui de reprendre la partie précédente non terminée, stockée dans un fichier texte. D'autre part, pour tout de même conserver cette notion de nombre de coups, ce nombre est affiché en fin de partie.

Pour ce qui est des options, nous avons bien programmé le roque et la promotion que nous avons envisagé plus tôt. De plus, nous sommes parvenus à ajouter la prise en passant et la mise en échec. Ces deux options supplémentaires n'étaient pas prévus mais nous ont tout de même paru fondamentales à ajouter à notre programme.

Nous n'avons pas jugé nécessaire de nous servir du cours d'infographie concernant l'affichage, puisqu'à l'aide des cours de l'année dernière, et à partir de l'utilisation de l'unité Crt, nous avons pu établir un rendu visuel tout à fait suffisant.

Ainsi, conformément aux attendus du cahier des charges nous rendons un jeu complet comprenant toutes les options de base. Nous avons parfois eu à revenir sur des idées d'implémentations que nous ne jugions pas pertinentes après réflexion et, au fur et à mesure des séances, nous avons pu constater que certaines fonctionnalités comme la mise en échec ou la prise en passant que nous estimions trop complexes étaient finalement envisageables et avaient une réelle importance pour un jeu d'échecs.

# II. Descriptif de la conception globale

Avant de commencer la description de la conception globale de notre projet, nous avons apporté quelques modifications à notre choix de structures de données initial. Nous les détaillons ci-dessous afin de faciliter la compréhension de ce qui va suivre. Dans un souci de clarté, l'analyse descendante ne comprend ni les entrées ni les sorties des sous-programmes. Ces dernières sont détaillées dans la partie descriptif, ainsi que le rôle et les explications des quelques modifications apportées à chacune de ces procédures ou fonctions.

# 1) Structure de données

Pour les besoins de la programmation, nous introduisons deux types « structure » qui permettent, pour le premier, de décrire la position d'une pièce sur l'échiquier (type de nom « Coordonnees ») et pour le second de décrire une case de l'échiquier (type de nom « CaseEchiquier ») à partir de la nature et de la couleur de la pièce qui l'occupe éventuellement. Cette couleur est une notion que nous précisons ci-après.

Type Coordonnees = Record

x, y : Integer ;

end ;

Type CaseEchiquier = Record

pion : String ;

c : T\_couleur ;

end ;

Pour encoder la couleur d'une case, nous introduisons un type énuméré de nom « T\_couleur » qui prend trois valeurs possibles : blanc ou noir lorsqu'une pièce occupe la case correspondante et neutre par défaut.

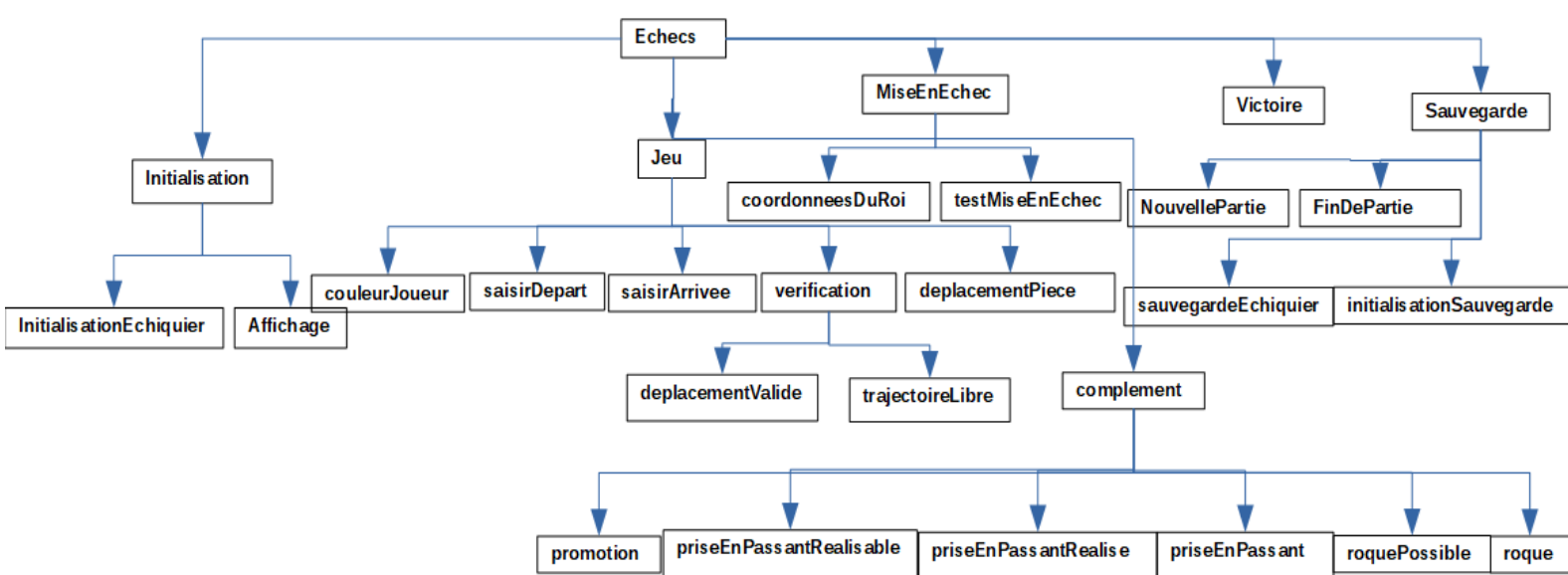
Type T\_couleur = (blanc, noir, neutre) ;

Enfin, le type T\_echiquier est un tableau à deux entrées composé d'items qui sont eux-mêmes de type « CaseEchiquier ».

Type T\_echiquier = Array [1..8,1..8] of CaseEchiquier ;

Dans notre première analyse globale, nous avons envisagé de nombreux autres types de données, qui après réflexion ne nous ont pas paru utile et ne faisaient qu'alourdir le programme. C'est pourquoi nous avons décidé de ne garder uniquement les types qui allaient réellement servir.

# 2) Analyse descendante



### 3) Descriptif détaillé des sous programmes

#### a) Programme principal

De manière à mieux comprendre les différentes unités et afin de contextualiser les différentes procédures et fonctions, nous vous présentons en premier lieu une description du fonctionnement de notre programme principal.

-Tout d'abord, le programme principal initialise les variables.

-Ensuite, le programme demande si le joueur souhaite débiter une nouvelle partie (nouvellePartie). Si le joueur commence une nouvelle partie alors on initialise l'échiquier (InitialisationEchiquier), et s'il souhaite continuer la partie précédente, alors, on initialise la sauvegarde (InitialisationSauvegarde).

-Avant que le tour de jeu débute, il faut attribuer de nouvelles valeurs à certaines variables : on attribue au joueur sa couleur (couleurJoueur), on ajoute un coup au compteur de coups, on vérifie quels roques sont encore réalisables (roquePossible), et on regarde si l'échiquier offre une possibilité de prise en passant (priseEnPassantRealisable).

-On affiche alors la couleur du joueur qui doit jouer, et on le prévient s'il est mis en échec (testMiseEnEchec).

-Le tour du joueur peut alors débiter. On lui propose tout d'abord (uniquement s'il en a la possibilité) de réaliser un roque (roque). Il peut aussi faire le choix de continuer de jouer (aJoue=false). Le joueur sélectionne alors une case de départ contenant l'une de ses pièces (saisirDepart), puis la case d'arrivée de sa pièce (saisirArrivée). A ce moment là, on se retrouve confronté, à deux possibilités, soit le joueur réalise une prise en passant (priseEnPassantRealise), soit il effectue un coup normal. En cas de coups normal, on doit vérifier que le déplacement suit les règles des échecs (déplacementValide) et qu'il n'y a pas de pièces empêchant ce déplacement (trajectoireLibre).

-On remplace alors, l'ancien échiquier par le nouveau qui prend en compte le déplacement (deplacementPiece ou priseEnPassant). De plus, si le joueur déplace un pion sur sa dernière ligne, le programme lui offre la possibilité de réaliser une promotion (promotion).

-L'échiquier, la couleur du joueur, la possibilité de réaliser les différents roques, ainsi que le nombre de coups joués sont alors sauvegardés dans un fichier texte (sauvegardeEchiquier). Le nouvel échiquier peut alors être affiché (affichage).

-Les tours de joueurs s'enchaînent ainsi jusqu'à ce qu'un joueur soit en échec à la fin de son tour. On affiche alors, la couleur du joueur gagnant et le nombre de coups dont il a eu besoin pendant 10 secondes, on remet alors à zéro la sauvegarde et on demande aux joueurs s'ils désirent faire une nouvelle partie (fin).

#### b) Unité initialisation

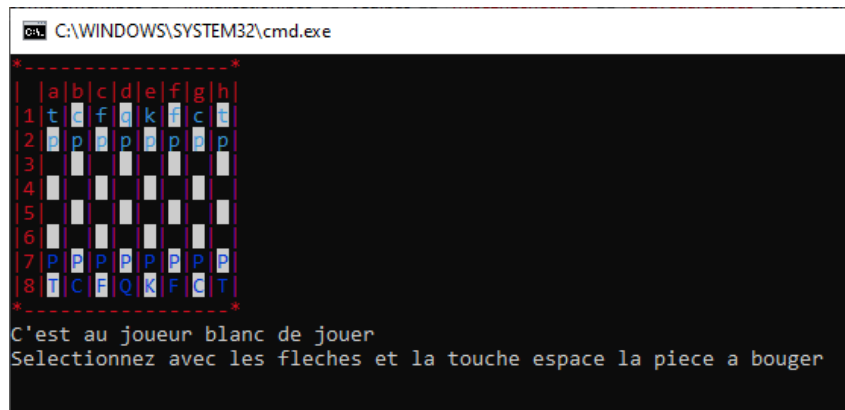
- i. procédure InitialisationEchiquier(S e: T\_echiquier);

Cette procédure initialise le jeu en affectant à chaque case de l'échiquier la pièce qui lui est associée en début de partie ainsi que sa couleur. C'est à dire que l'on affecte à chaque case de l'échiquier la lettre qui correspond à la pièce qui l'occupe (une lettre minuscule pour les blanches et majuscule

pour les noires), et une couleur. De plus, lorsqu'une case est inoccupée on lui affecte un espace comme valeur de pièce et une couleur neutre.

ii. procédure Affichage(E e: T\_echiquier);

Cette procédure permet d'afficher l'échiquier « e » à l'écran. Cet affichage est textuel et en couleur. En effet, la procédure affiche les pièces de la couleur du joueur et met en forme l'échiquier avec un damier.



### c) Unité jeu

i. fonction couleurJoueur(joueur: T\_couleur): T\_couleur;

Cette fonction permet, à partir de la valeur de la couleur donnée en entrée, de restituer en sortie la couleur opposée.

ii. procédure saisirDepart (E e: T\_echiquier; E joueur : T\_couleur ; S coordD: Coordonnees ; S x,y : Entier);

Cette procédure prend en entrée l'échiquier et la couleur du joueur qui doit jouer. Lors de l'exécution de celle-ci, le joueur saisit la pièce à déplacer à l'aide des flèches et de la barre d'espace. La procédure retourne alors la position sur l'échiquier de la pièce à déplacer, à condition que le joueur choisisse bien une de ses pièces. De plus, la procédure renvoie les coordonnées (x,y) du curseur.

iii. procédure saisirArrivee (E e: T\_echiquier; E joueur:T\_couleur; E x,y: Integer ; S coordA: Coordonnees);

Cette procédure prend en entrée l'échiquier et la couleur du joueur qui doit jouer ainsi que les coordonnées (x,y) du curseur. Lors de l'exécution de celle-ci, le joueur saisit, à partir de la pièce à déplacer, la case d'arrivée. La procédure retourne alors sa position sur l'échiquier, à condition que le joueur choisisse bien une case neutre ou contenant une pièce adverse.

iv. fonction deplacementPiece(e: T\_echiquier;coordD, coordA: Coordonnees):T\_echiquier;

Cette fonction permet, à partir de la donnée d'un échiquier et des positions de départ et d'arrivée, de retourner un nouvel échiquier résultant du déplacement de la pièce. On affecte à la case d'arrivée les propriétés de la case de départ (rôle et couleur de la pièce de départ). La case de départ prend alors une valeur neutre.

## d) Unité vérification

- i. fonction `deplacementValide(e: T_echiquier; joueur: T_couleur; coordD, coordA: Coordonnees): booléen;`

A partir de la donnée d'un échiquier, d'une position de départ et d'une position d'arrivée, cette fonction retourne un booléen qui caractérise la validité (ou non) du déplacement proposé en fonction de la nature de la pièce, de la couleur du joueur et du déplacement rentré. Cette fonction vérifie que la pièce suit bien ses règles de déplacement propre (tour en ligne droite, fou en diagonal, etc.). Cette validité est partielle et complétée par la fonctionnalité qui suit.

- ii. fonction `trajectoireLibre(e: T_echiquier; coordD, coordA: Coordonnees): booléen;`

A partir de la donnée d'un échiquier, d'une position de départ et d'une position d'arrivée, cette fonction retourne un booléen qui caractérise la présence ou non d'un obstacle sur la trajectoire qui relie les deux positions.

## e) Unité score

- i. fonction `Victoire(e: T_echiquier): booléen;`

A partir de la donnée d'un échiquier, cette fonction teste l'absence (ou non) de l'un des deux rois sur le plateau. Cette absence caractérise une fin de partie.

## f) Unité complément

- i. procédure `roquePossible (E e: T_echiquier; S petitRoqueBlancPossible, petitRoqueNoirPossible, grandRoqueBlancPossible, grandRoqueNoirPossible: Booléen);`

A partir de la donnée d'un échiquier, cette procédure s'assure que les tours et rois n'ont pas été déplacés depuis le début du jeu. La procédure renvoie alors, pour chaque roque (petit roque pour le joueur blanc, grand roque pour le joueur blanc, petit roque pour le joueur noir, et grand roque pour le joueur noir) s'il est encore réalisable.

- ii. procédure `roque (E joueur: T_couleur; E petitRoqueBlancPossible, petitRoqueNoirPossible, grandRoqueBlancPossible, grandRoqueNoirPossible: Boolean; E/S e: T_echiquier; E/S aJoue: booléen);`

Cette procédure exploite la donnée d'un échiquier, les résultats du test de la procédure précédente ainsi qu'une variable d'état globale « aJoue » pour procéder au roque. Lors de l'exécution de cette procédure, il est proposé au joueur jouant son tour de réaliser un roque si le roque est toujours possible (voir procédure précédente) et que les cases entre le roi et la tour concernées sont vides. Le cas échéant, un menu apparaît et le joueur choisit avec les flèches et la barre d'espace ce qu'il compte faire.

- iii. procédure `promotion (E coordA: coordonnees; E joueur: T_couleur; E/S e: T_echiquier);`

A partir de la donnée d'un échiquier, d'une position d'arrivée et de la couleur du joueur, cette procédure commence par s'assurer que la pièce concernée est un pion, puis que celui-ci se rend sur « sa » dernière ligne du plateau de jeu (ligne 8 pour les pions blancs et 1 pour les pions noirs). Le joueur a alors le choix de changer son pion en une dame, un cavalier, un fou ou une tour.

- iv. procédure priseEnPassantRealisable (E coordD, coordA: coordonnees; E joueur:T\_couleur;  
E e: T\_echiquier; S coordPriseEnPassant:coordonnees; S priseEnPassantPossible: booléen);

A partir de la donnée d'un échiquier, d'une position de départ et d'arrivée ainsi que de la couleur du joueur, cette procédure vérifie si la prise en passant est possible, c'est-à-dire si le pion déplacé de deux cases arrive à coté d'un pion adverse au cours de son déplacement. En complément, cette procédure retourne la position du pion adverse si le joueur adverse décidait de procéder à la prise en passant (c'est à dire la position derrière le pion ayant avancé de deux cases).

- v. fonction priseEnPassantRealise(priseEnPassantPossible: boolean;  
coordD,coordA,coordPriseEnPassant: coordonnees;joueur:T\_couleur;  
e:T\_echiquier):booléen;

A partir de la donnée d'un échiquier, d'une position de départ et d'arrivée, de la couleur du joueur et du résultat de la procédure précédente, cette fonction renvoie un booléen qui indique si le joueur a réalisé une prise en passant, c'est à dire si les coordonnées de déplacement correspondent à un pion et coïncide avec les coordPriseEnPassant.

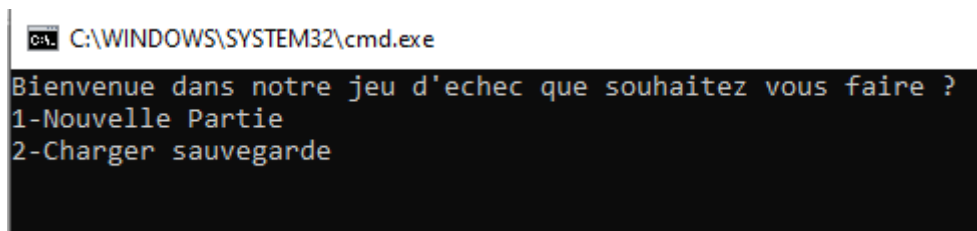
- vi. fonction priseEnPassant (e: T\_echiquier; joueur:T\_couleur; coordD, coordA: Coordonnees):  
T\_echiquier;

A partir de la donnée d'un échiquier, d'une position de départ et d'arrivée ainsi que de la couleur du joueur, on réalise une prise en passant (suppression du pion adverse et déplacement du pion du joueur concerné).

## g) Unité sauvegarde

- i. fonction nouvellePartie():booléen;

Cette fonction propose un menu à deux choix permettant soit de reprendre la partie sauvegardée soit d'en démarrer une nouvelle. Elle retourne le booléen false dans le premier cas et true sinon.



```
C:\WINDOWS\SYSTEM32\cmd.exe
Bienvenue dans notre jeu d'echec souhaitez vous faire ?
1-Nouvelle Partie
2-Charger sauvegarde
```

- ii. fonction finDePartie():booléen;

Cette fonction propose un menu à deux choix demandant à l'utilisateur s'il souhaite - ou non - continuer à jouer. Dans le premier cas, il retourne false et dans le second true.

- iii. procedure sauvegardeEchiquier(E e: T\_echiquier; E joueur:T\_couleur; E  
petitRoqueBlancPossible,petitRoqueNoirPossible,grandRoqueBlancPossible,grandRoqueNo  
irPossible: boolean ; E coups:Integer);

Cette procédure permet de sauvegarder la partie en cours dans un fichier texte. En plus de sauver la disposition des pièces sur l'échiquier, on inscrit la couleur du joueur dont c'est le tour de jouer, ainsi que la possibilité pour chacun des camps de roquer avec 4 mots (qui sont alternativement



« possible » ou « impossible » selon que le roque correspondant est encore possible ou non). De plus on sauvegarde le nombre de coups déjà joués.

- iv. `procedure initialisationSauvegarde(S e: T_echiquier; S joueur:T_couleur; S petitRoqueBlancPossible,petitRoqueNoirPossible,grandRoqueBlancPossible,grandRoqueNoirPossible: Boolean ; S coups: Integer);`

Cette procédure permet de restaurer une partie en cours en la chargeant à partir de la donnée du fichier texte de sauvegarde, de nom « sauvegarde.txt ».

## **h) Unité mise en échec**

- i. `procédure coordonneesDuRoi(E e: T_echiquier; E joueur:T_couleur ; S coordR: Coordonnees) ;`

A partir de la donnée d'un échiquier et de la couleur du joueur, cette procédure retourne la position du roi du joueur sur l'échiquier.

- ii. `fonction testMiseEnEchec(e: T_echiquier; joueur:T_couleur): booléen;`

A partir de la donnée d'un échiquier, de la position du roi et de la couleur du joueur, cette fonction retourne un booléen qui précise si le roi du joueur concerné est, ou non, mis en échec par l'une des pièces du joueur adverse.

## **III. Guide d'utilisation du programme**

Notre jeu d'échec est plutôt intuitif. En effet, il suffit de lire les indications affichées sur le terminal. Pour déplacer le curseur, le joueur appuie sur les flèches directionnelles et il sélectionne à l'aide de la barre d'espace. Le fonctionnement est identique dans les menus mais aussi en jeu.

Ainsi à chaque coup, la couleur du joueur qui doit jouer s'affiche. Il sélectionne alors la pièce à déplacer en promenant le curseur sur l'échiquier à l'aide des flèches directionnelles, puis il enfonce la barre d'espace afin de valider son choix. Il lui faut alors se déplacer par le même moyen sur la case où il souhaite emmener sa pièce. S'il sélectionne un coup qui n'est pas autorisé par les règles officielles du jeu, le joueur est alors obligé de rentrer de nouvelles coordonnées.

On pourra noter que, le camp blanc est matérialisé par les pièces en minuscule et le camp noir matérialisé par les pièces en majuscules.

## **IV. Retour sur le travail de groupe**

Ce projet nous a permis d'identifier plusieurs difficultés auxquelles nous avons du faire face. Tout d'abord l'implémentation de ce code nous a demandé beaucoup de temps car nous n'étions, au début, pas forcément très à l'aise en informatique. Pour faciliter notre apprentissage, nous nous sommes souvent retrouvés pour travailler ensemble car nous étions plus efficaces que lorsque nous codions séparément. De plus, n'ayant pas forcément la même manière de voir les choses, le fait de débattre nous permettait de trancher quant à la solution la plus efficace. Le confinement n'a ainsi pas simplifié ce développement. En effet, coder de manière individuelle

rendait en général le travail plus difficile car nous nous perdions rapidement dans les noms affectés aux types et variables.

Nous n'avons pas pour autant arrêté de travailler ensemble, puisque nous avons pu trouver des alternatives telles que discord qui nous a permis d'échanger à distance. Nous avons pu, à l'aide de partage d'écrans, alterner « le codeur ». En général, l'un de nous trois rédigeait et nous pouvions ainsi échanger en temps réel. Cette manière de faire s'est finalement révélée être un bon compromis.

Cependant, nous avons tout de même ressenti l'absence d'un certain nombres d'outils qui nous auraient facilité le travail collectif : convention de nommage partagée, dépôt de source partagé, et gestion de versions. Rétrospectivement, notre mode de fonctionnement était perfectible et il semblerait que l'utilisation d'un outil comme Git aurait pu répondre à certaines des limites que nous avons identifiées. Nous veillerons pour les prochains projets à adopter un tel outil.

Enfin, afin de suivre notre avancement et d'être sûrs de respecter les délais que nous nous étions fixés, nous avons tenu un journal décrivant l'avancement du projet. Nous avons finalement été plus rapides que prévu, ce qui nous a permis d'apporter quelques améliorations à notre programme.

Pour faciliter la lecture de ce journal, nous avons choisi d'y inscrire en italiques les noms de procédures et fonctions :

#### **21.09.2020**

- création v1 (on suit l'analyse descendante au type près)
- unit TypesDeDonnees.pas (cf. diaporama de présentation)
- unit Initialisation.pas (*creationEchiquier*, *creationPiece*)
- unit Affichage.pas (affichage non pas d'un échiquier mais simplement de colonnes vérifiant certaines conditions -cases libres? roque possible?-)
- ProgrammePrincipal.pas

#### **01.10.2020**

- création v2 (on reprend la structure générale car nous étions bloqués au niveau de l'affichage : changement des structures de données et quelques modifications apportées à l'analyse descendante)
- unit Initialisation.pas (*InitialisationEchiquier*, *Affichage*)
- unit Jeu.pas (*saisirDepart*, *saisirArrivee*, *deplacementPiece*)
- unit Score.pas (*Victoire*)
- ProgrammePrincipal.pas (partie jouable sans aucune vérification : on annonce la fin du jeu)

#### **02.10.2020**

- unit Jeu.pas (*couleurJoueur*) : permet l'alternance des joueurs dans le programme principal
- unit Verification.pas (*deplacementValide*)
- ProgrammePrincipal.pas (partie jouable sur échiquier coloré - modification d'*Affichage* , avec quelques vérifications : on annonce la fin du jeu)

#### **09.10.2020**

- unit Verification.pas (*trajectoireLibre*)
- ProgrammePrincipal.pas (partie jouable avec vérifications : on annonce le gagnant)

#### **15.10.2020**

- unit Sauvegarde.pas (*nouvellePartie, sauvegardeEchiquier, initialisationSauvegarde*)
- ProgrammePrincipal.pas (le joueur peut choisir s'il souhaite, ou non, recommencer une nouvelle partie)

#### **22.10.2020**

- Amélioration de notre code : simplifications de certaines procédures, amélioration du rendu visuel, mise en forme plus lisible de notre code.

#### **03.11.2020 et 04.11.2020**

- unit complement.pas (*roquePossible, roque*)

#### **11.11.2020**

- unit complement.pas (*promotion*)

#### **27.11.2020**

- unit complement.pas (*priseEnPassant*)
- Rédaction de la section « Descriptif de la conception globale » du rapport

#### **02.12.2020 et 04.12.2020**

- unit MiseEnEchec.pas (*testMiseEnEchec, coordonneesDuRoi*)
- Rédaction de la section « Retour sur le travail de groupe » du rapport

#### **05.12.2020 et 06.12.2020**

- On reprend notre programme et on le restructure.
- Fin de la rédaction du dossier.
- On entame la préparation du support de présentation orale.

## **Conclusion**

En premier lieu, ce projet de jeu d'Échecs nous a permis d'améliorer considérablement notre niveau en programmation. C'est ensuite une expérience tout à fait enrichissante du point de vue du travail en équipe. Ainsi, nous avons appris à confronter nos idées et à débattre mais aussi à nous organiser et à respecter des délais.

L'exercice qui nous a été demandé nous sera utile dans le cadre de notre futur métier. En effet, un ingénieur doit avoir un profil complet afin de pouvoir aisément s'adapter à toutes formes de situations. Il doit pouvoir naviguer entre différents secteurs d'activités et ce, de manière rapide et efficace, afin d'être toujours plus productif. Faire preuve de productivité nécessite d'être en mesure de travailler en équipe. Il lui est impératif de savoir communiquer avec ses collaborateurs et de comprendre « l'autre », ses manières de travailler, mais aussi ses opinions et convictions propres. C'est pourquoi, au-delà d'avoir un aspect technique tout à fait intéressant, programmer ce jeu a été un bon moyen pour nous d'apprendre à nous écouter et à partager nos savoir-faire.