GATEL Kévin
MORIN Simon
SALLÉ Pascale

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet Informatique

Jeu d'échecs

2019/2020

STPI2 (Gr C)

Sommaire

Introduction

Le principe de ce projet est de nous faire coder un jeu ou une application par exemple que l'on choisit. Il nous faut alors établir une première version d'analyse descendante – pouvant évidemment être modifiée tout au long du projet – ainsi qu'un cahier des charges présentant le concept et les fonctions utiles du programme, correspondant à la conception globale de notre programme. Puis dans un deuxième temps, de réaliser la conception détaillée du projet choisi, donc l'écriture, ici en Pascal, du code. Tout cela en respectant la limite de temps qui nous était donnée. Il nous a donc fallu les différentes tâches parmi les membres du groupe en prenant en compte les capacités de chacun.

Pour ce projet nous avons décidé de programmer un jeu d'échecs, en partant du principe que l'on faisait jouer deux joueurs une partie d'échecs classique, en se laissant la possibilité de rajouter des options en cours d'écriture.

Pour expliquer plus en détails le déroulement du projet, nous commencerons par voir en quoi consiste exactement ce que l'on peut appeler une partie d'échecs classique, ainsi que les différents coups spéciaux qui ont été ajoutés dans le programme, au fur et à mesure de son écriture.
Ensuite, nous verrons les différentes attentes liées à la conception de ce jeu d'échecs, de même que les types/structures utilisées au travers du cahier des charges et des différentes versions de l'analyse descendante.
Puis, dans un troisième temps, nous analyserons le programme en lui-même, en s'attardant sur certaines fonctions ou procédures particulières.
Enfin, nous aborderons les différents problèmes que nous avons dû contourner pour que tout s'exécute correctement.

I. Présentation des échecs et certains coups spéciaux

A. Le jeu d'échecs

Avant d'expliquer le programme et les différentes fonctionnalités nécessaires pour jouer une partie d'échecs, il faut d'abord connaître les règles de base d'une partie d'échecs dits « classiques ».
Ce jeu se joue sur un échiquier comportant 64 cases alternant le blanc et le noir, sachant que la première case en haut à gauche est toujours blanche. Il se joue à deux joueurs qui ont chacun 16 pièces qui leur sont attribuées – 2 tours, 2 cavaliers, un roi, une reine, 2 fous et 8 pions. Les joueurs déplacent leurs pièces chacun leur tour, lorsqu'une pièce – disons noire – prend la place d'une blanche une fois son déplacement effectué, la pièce blanche est « prise », et est sortie du jeu. Chaque pièce a un déplacement qui lui est propre :

- Les fous se déplacent en diagonale ;
- Les cavaliers en L ;
- Les tours en ligne droite ;
- Les pions droit devant lui et seulement d'une case à la fois, sauf lors de son premier déplacement où il peut être bougé de une ou deux cases ;
- La reine dans toutes les directions (lignes droites ou diagonales) ;
- Et le roi d'une seule case peu importe la direction.

Le but des échecs est de « mettre en échec » le roi adverse, à ce moment-là le joueur adverse n'a qu'un nombre limité de déplacement possible puisqu'il est obligé de protéger son roi à tout prix. La partie se termine si le roi d'un des deux joueurs est « échec et mat », c'est-à-dire qu'il n'a aucun moyen de protéger son roi face à l'adversaire. Cela dit, un « échec et mat » n'est pas le moyen de terminer une partie, puisqu'une partie ne mène pas nécessairement à la victoire d'un des joueurs – on parle alors de partie nulle, quand aucun joueur ne peut conclure.

B. Les coups spéciaux

Aux échecs il est possible de ne pas pouvoir conclure une partie : c'est le cas du pat. Lorsque le joueur devant effectuer un déplacement n'en a aucun autorisé, sous peine de mettre son propre roi en échec – il ne peut jouer – on dit alors qu'il est pat, et la partie est alors nulle (aucuns vainqueurs, aucuns perdants). C'est pour ça que notre programme a dû prendre en compte cette particularité du jeu pour pouvoir être complet, on a dû implémenter une fonctionnalité qui permet de savoir si le joueur est pat.

Outre, le pat, deux coups spéciaux sont autorisés en plus des déplacements légaux : le roque et la prise en passant. Le premier consiste à permettre au joueur de déplacer le roi de 2 ou 3 cases d'un coup, de déplacer en une seule fois une tour et le roi, et ainsi de pouvoir faire sauter la tour par-dessus une autre pièce – le roi. Il s'agit d'un bon moyen pour sécuriser son roi en l'éloignant du centre, mais aussi de pouvoir recentraliser rapidement une tour. Le roque est un type de déplacement de base mais pour être effectué il y a certaines conditions à respecter : tout d'abord il faut qu'il n'y ait aucune pièce présente entre le roi et la tour, ensuite il faut que ces 2 pièces n'aient pas été déplacées jusqu'à l'instant où le joueur souhaite utiliser le roque – donc que les 2 pièces soient sur leur case d'origine. De plus, le roi ne doit ni être en échec, ni devoir passer par une case sur laquelle le roi pourrait être échec.

Enfin, le deuxième coup évoqué, la prise en passant, correspond à un coup particulier pour prendre un pion adverse. Si un joueur a son pion sur la 5ème rangée et que son adversaire fait bouger son pion de 2 cases en avant – ce qui n'est autorisé que s'il s'agit du premier déplacement de ce pion – dans

une colonne voisine, les pions sont alors côte à côte sur la même ligne et le premier pion peut prendre le second. Pour pouvoir prendre cette pièce, le joueur doit déplacer son pion en diagonale sur la 6ème rangée et la colonne du pion qu'il veut prendre.

II. Les attentes du projet

Concernant les attentes de ce projet, le but principal était bien sûr de coder un jeu d'échecs pour que deux joueurs puissent jouer l'un contre l'autre. Pour établir clairement ce qu'il fallait faire pour mener à bien ce projet, nous avons d'abord établi un cahier des charges dans lequel on retrouve une liste des fonctionnalités nécessaires prioritaires .

A. Le cahier des charges

Un premier point essentiel était d'afficher le plateau à son état initial – soit sans déplacement effectué, il correspond à un tableau à 2 dimensions de caractères, chaque pièce est représentée par une lettre pour la facilité d'affichage. Dans cette continuité, il fallait que notre code actualise ce plateau à chaque déplacement d'une pièce par un joueur, mais avant de pouvoir le mettre à jour, il était important que la validité du déplacement demandé soit vérifiée. Pour cela, il nous fallait des procédures pour chaque type de mouvement puisque les pièces n'ont pas toutes la même mobilité autorisée, comme évoqué précédemment. Permettre l'Interaction Homme-Machine (IHM) était également primordial puisque le jeu devait être interactif pour les utilisateurs, et surtout que le programme soit facile à utiliser par ces derniers. Enfin, le programme devait pouvoir décréter un vainqueur, donc être capable de vérifier si un roi est ou non en échec, et par le même principe, s'il est ou non en échec et mat pour terminer la partie. Une obligation du cahier des charges – car imposée par les consignes – était la présence d'un fichier que l'on utiliserait dans le code. Toutes ces contraintes ont été mises en évidence dans la phase d'étude du problème – à savoir de quoi avons-nous besoin pour que le jeu fonctionne.
Ce premier cahier des charges a été par la suite complété ou simplement modifié au fur et à mesure de l'écriture du code (cf. Annexe 1).

B. L'analyse descendante

Après ce premier bilan avec l'écriture du cahier des charges, nous avons alors réalisé une première version de l'analyse descendante du programme (cf. Annexe 2). Ce premier jet représente la structure globale du programme, on y retrouve notamment une « case » par procédure de vérification de validité de déplacement des pièces – une pour chaque pièce – ainsi que les procédures de vérification de l'échec et de l'échec et mat. Concernant la case appelée « Protection Roi apr. Dpt », son but est de contrôler que le roi soit toujours bien protégé après le déplacement d'une pièce, elle permet de s'assurer que le roi du joueur qui est en train de jouer ne sera pas mis en échec avec le déplacement qu'il souhaite effectuer. La première modification qui a été apportée à cette analyse descendante est celle d'avoir en quelque sorte regroupé – ou bien remis les deux sous un même nom – la procédure « Protection Roi apr. Dpt » et « Vérification Échec », car en réalité elles ont le même objectif final.

La deuxième version de l'analyse descendante (cf. Annexe 3), qui correspond à la version définitive de notre programme, a été non seulement modifiée par rapport à celle de départ, mais également complétée puisque des fonctionnalités supplémentaires ont été rajoutées au programme : le roque, le pat et la prise en passant en font partie. Il est également à noter qu'un Timer/minuteur a été ajouté, pour limiter le temps de jeu des joueurs – chacun a 10 minutes de jeu.

Pour commencer, concernant l'initialisation, la procédure du même nom remplie le tableau de pièces case par case (sans oublier les cases « vide »), cette initialisation donne la configuration initiale du plateau, lorsque l'on recommence une nouvelle partie. Une fonctionnalité ajoutée est la procédure « Save » qui permet de sauvegarder dans le fichier « sauve.txt » la disposition des pièces en fonction du joueur en cours, des possibilités de roque, de la situation d'échec ou non du joueur, du temps de

jeu écoulé (cf. Timer) et enfin d'un compteur. Cette procédure pourra alors être appelée par la procédure « Load » qui, comme son nom l'indique, charge la sauvegarde « sauve.txt », et permet ainsi aux joueurs de pouvoir reprendre une partie non terminée.

Ensuite, par rapport à l'affichage de l'échiquier, cette procédure possède deux sous-procédures, au lieu d'une seule comme c'était le cas en début de conception. En effet, elle appellera la procédure « Damier » qui elle-même affichera une image (noire ou blanche) et est repérée par ses coordonnées cartésiennes (x ; y) – avec l'axe des y qui augmente en descendant et celui des x augmentant lorsque l'on se déplace vers la droite. « Damier » permet aussi de savoir de quelle couleur est la case en s'appuyant sur le fait que si la somme (x+y) est paire alors la case est noire, et si elle est impaire alors la case est blanche. Ensuite la procédure d'affichage appelle « Damier » pour afficher l'échiquier ainsi que toutes les pièces présentes sur le plateau en parcourant toutes les cases du tableau de pièces.
En ce qui concerne le Timer évoqué plus tôt, il permet de calculer et d'afficher le temps qu'il reste à jouer pour chacun des joueurs. Lorsqu'un tour de jeu est en cours la procédure « Timer » est appelée dans chaque boucle pour que le temps défile à chaque instant du programme.

Maintenant à propos de la procédure « Dpt Valide » soit « déplacement valide », celle-ci n'appelle pas plusieurs procédures comme c'était le cas dans la première analyse descendante, mais ne forme qu'une seule procédure (pour plus de détails cf. partie III.A.). D'autre part, toujours par rapport aux déplacements, deux procédures de déplacement ont été créées : la première, « Dpt » est appelée lorsque le joueur a choisi sa pièce avec le curseur, après cela le joueur déplace le curseur pour sélectionner la case d'arrivée de sa pièce. A ce moment-là, la procédure « Dpt2 » est lancée, on vérifie alors si le déplacement est correct, s'il crée ou non une situation d'échec, on modifie les variables de la procédure « Roc », etc. Ici, si une des conditions n'est pas respectée, alors le coup est annulé et on demandera au joueur de rejouer. En ce qui concerne l'affichage des déplacements, la procédure « Affichage Dpt » permet d'indiquer au joueur, après qu'il est sélectionné sa pièce, de colorer les cases sur lesquelles il peut déplacer cette pièce avec un code couleur spécifique :

- En rouge, celles où l'on peut se déplacer sans spécificité ;
- En bleu, celles où l'on peut « manger » une pièce adverse ;
- En violet, celles impliquant la prise en passant ou le roque ;
- En vert, celles où le roi du joueur se retrouvera en échec (parce que cette pièce protégeait le roi ou bien que le déplacement ne permet pas de protéger le roi alors qu'il est échec).

Par ailleurs, les coups spéciaux (soit le roque, le pat et la prise en passant) possèdent chacun leur procédure. Premièrement, celle du roque sert à vérifier si ce coup est possible, ainsi on vérifie si les rois et tours n'ont pas changé de position – pour cela on leur attribue des valeurs lorsqu'ils bougent. On s'assure également qu'aucune autre pièce n'est présente entre la tour et le roi, et enfin que ce dernier n'est pas en situation d'échec avant et après le roque. Deuxièmement, pour la prise en passant, la procédure sert uniquement à autoriser ou non la prise en passant pour les pions. Et dernièrement pour le pat, la procédure associée appellera « Dpt Valide » et « Echek », et regardera si les joueurs ont effectué plusieurs fois les mêmes coups. De là le jeu sera pat, et donc comme évoqué précédemment le jeu se terminera sans qu'il n'y ait de vainqueur.

Pour terminer, les procédures d'échec (donc « Echek » et « EchekMat ») consistent pour la première à prendre les coordonnées du roi et a regardé s'il se trouve sur la trajectoire d'une pièce ennemie – à noter qu'il n'est alors pas nécessaire de prendre en compte toutes les pièces du jeu. Enfin, pour la deuxième, on suppose initialement qu'un joueur est effectivement échec et mat, puis la procédure va appeler « Dpt Valide » et « Echek » pour réfuter cette hypothèse si celle-ci est fausse.

C. Les types utilisés

Pour l'écriture du programme un seul type a été utilisé car l'ensemble du codage reposait essentiellement sur des mathématiques et de la logique. De fait l'unique type utilisé est le tableau à 2 dimensions contenant les pièces et les « vides » que composent l'échiquier. Il s'agit donc d'un tableau de chaîne de caractères (string) car chaque pièce est symbolisée par une lettre, et le plateau comporte des cases de « vide ».
A noter qu'aucune structure n'a été utilisée non plus.

III. Le programme

Dans cette partie, nous avons décidé d'expliquer les trois procédures les plus essentielles de notre programme, à savoir « Dpt Valide », « Echek » et « EchekMat ».
Avant de débuter l'explication, il est important de notifier qu'il suffit d'expliquer uniquement la moitié de chaque procédure pour comprendre leur fonctionnement global. En effet, comme deux joueurs s'affrontent avec chacun sa couleur de pièces, il n'est nécessaire de détailler qu'un des deux pour comprendre le reste, puisque le principe utilisé sera exactement le même pour les deux joueurs, seules certaines valeurs seront différentes entre les deux puisque leurs pièces ne démarrent pas du même endroit sur le plateau.
De plus, le système de coordonnées est le même pour tout le programme : l'axe des x, en abscisse, augmente de gauche à droite, l'axe des y, en ordonnée, augmente du haut vers le bas, et l'origine est la case (1 ;1). Les pièces blanches partent du bas du plateau donc des valeurs y=7 ou 8.

A. Déplacement valide

Pour cette procédure, il faut vérifier les déplacements au cas par cas (fou, roi, etc.). Cependant, une condition peut être vérifiée hors d'un déplacement d'une pièce en particulier : les coordonnées de la case d'arrivée (x2tab ;y2tab) doivent être différentes de celles de départ (xtab ;ytab). Si ce n'est pas le cas, alors le déplace est invalide, et on demandera au joueur de rejouer – ceci est valable pour tous les cas où le déplacement sera jugé invalide.

Tout d'abord commençons par le pion, il y a trois cas à considérer pour pouvoir bouger cette pièce : le premier est la cas où le pion n'a encore jamais été déplacé. Ici, il faut avant tout s'assurer que le pion est bien sur sa « ligne de départ », mais également être sûr qu'il va bien se déplacer sur la droite x=xtab=x2tab. Puis, soit le joueur souhaite avancer de 2 cases et on vérifie que les 2 cases correspondantes sont bien vides, soit il souhaite se déplacer d'une seule case et il suffit alors de vérifier que cette case est vide.
Le deuxième cas est celui où le pion a déjà bougé sur le plateau, on s'assure qu'il n'y a bien qu'une case d'écart entre la position y initiale et la finale, et de nouveau que la case souhaitée n'est pas déjà occupée.

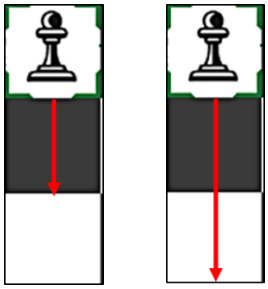
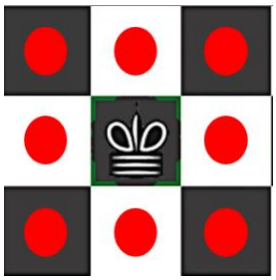


Figure 1 : Déplacement du pion

Le dernier est le cas de la prise d'une autre pièce. Ici c'est une vérification légèrement différente car il faut s'assurer que la position finale en y est bien égale à celle de départ moins 1 (y2tab=ytab-1) – comme dans le deuxième cas – mais aussi que la position x d'arrivée correspond à celle du départ +/- 1 (x2tab=xtab+/-1). Cette fois-ci il faut que la case contienne ennemie pour que le déplacement soit valide.
Si toutes les conditions sont respectées, alors le déplacement du pion est autorisé.

Figure 2 : Prise d'une pièce



Poursuivons avec le déplacement du roi, pour lui il faut vérifier que les distances (donc des valeurs absolues) x2tab-xtab et y2tab-ytab sont bien inférieures ou égales à 1, et ceci pour être sûr qu'on ne le fait pas se déplacer plus loin que le carré de cases autour de lui. A cela il faut rajouter le fait de vérifier que les soustractions xtab-x2tab et ytab-y2tab sont différentes de 0, puisque la pièce doit se déplacer, et enfin que la c ase souhaitée ne contienne pas de pièce alliée. Ainsi on pourra déclarer que le déplacement est correct.

Figure 3 : Déplacement du roi

Une autre pièce avec peu de vérifications est le cavalier – pouvant se déplacer en L. En premier lieu, on vérifie que la somme des distances (x2tab-xtab) et (y2tab-ytab) vaut bien 3, cependant cela ne suffit pas pour être certain de se déplacer en L. Pour cela, il faut s'assurer que ces distances sont différentes de 0, puisque si elles valent 0 ça signifierait que le cavalier se déplacerait en ligne droite (comme une tour ou une reine) ce qui est impossible.

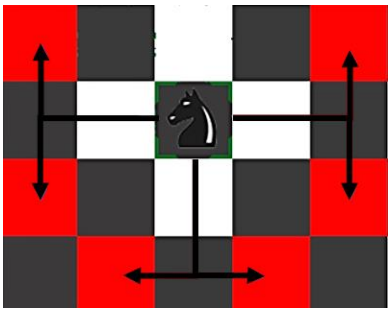


Figure 4 : Déplacement du cavalier

Puis on part du postulat que le déplacement est bon, et on cherche à savoir s'il est mauvais en regardant si la case d'arrivée contient une pièce alliée, si c'est le cas alors le postulat de départ est réfuté et le déplacement est invalidé.

Pour les déplacements des trois pièces restantes, deux variables vont être utilisées : « casepleine » et « constante » toutes les deux initialisées à 0. La valeur de la première changera si une pièce ennemie se trouve dans la case souhaitée et prendra la valeur 1, tandis que la deuxième prendra la valeur 1 pour permettre au programme de sortir des boucles. En effet, comme la tour, le fou et la reine n'ont en quelque sorte pas de limite de nombre de cases lorsqu'ils se déplacent (ce nombre peut être aussi bien 1 que 4), du coup il faut utiliser des boucles. De ce fait pour pouvoir sortir de la boucle, il a fallu introduire une constante qui changeait de valeur pour pouvoir effectuer cette sortie.

Débutons par la tour, pour cette pièce on peut distinguer deux types de déplacements : verticaux et horizontaux.

Pour les déplacements verticaux, il est essentiel que la tour se déplace sur une même droite x=xtab=x2tab. Ensuite on peut de nouveau distinguer 2 cas :

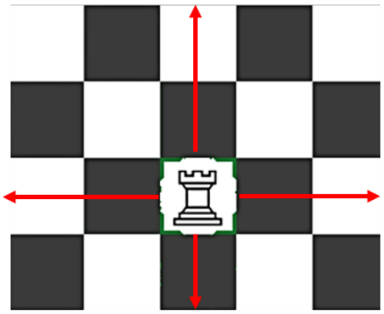


Figure 5 : Déplacement de la tour

- Les déplacements hauts : donc lorsque la position y finale est plus grande qu'initialement dans le cas d'une tour blanche et l'inverse pour une tour noire. On lance une boucle allant de y2tab à ytab+1, puisqu'on ne prend pas la valeur correspondant à ytab vu que le désire changer de case. Cette boucle va vérifier si les cases de cette trajectoire contiennent des pièces ennemies alors casepleine=0, s'il s'agit de pièces alliées en revanche casepleine=1.
- Les déplacements bas : donc lorsque y2tab<ytab pour les pièces blanches et inversement pour les pièces noires. Ici, même principe mais cette fois la boucle ira de y2tab à ytab-1 car on « descend » l'axe y, et encore une fois on ne souhaite pas rester sur place.

Pour les déplacements horizontaux, cette fois la tour se déplace sur une droite d'équation y=ytab=y2tab, et on a toujours 2 cas :

- Les déplacements vers la droite : soit lorsque x2tab>xtab, la boucle va de x2tab à xtab+1 (pour ne pas prendre la case d'origine de la tour en compte) peu importe la couleur des pièces, puis le même raisonnement que pour les déplacements verticaux est déroulé
- Les déplacements vers la gauche : lorsque x2tab<xtab, la boucle va de x2tab à xtab-1 puis de nouveau la même chose s'applique.

Dans tous les cas, peu importe le type de déplacement en ligne droite demandé, le déplacement sera jugé valide uniquement si casepleine=0 en sortie de boucle.

Continuons avec le fou, pour ce dernier il est tout d'abord nécessaire de calculer deux distances c1 :=abs(xtab-x2tab) et c2 :=abs(ytab-y2tab). Une fois cela fait, on commence par regarder si ces

valeurs sont distinctes ou égales, si elles sont distinctes alors casepleine=1, si elles sont égales en revanche on peut, comme pour la tour, séparer les déplacements diagonaux en 4 cas :

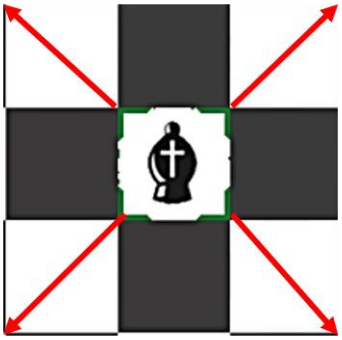


Figure 6 : Déplacement du fou

- La diagonale droite haute : la boucle va de la ytab-y2tab jusqu'à 1, et pas jusqu'à 0 puisque cela reviendrait à dire que les deux positions sont confondues, or c'est impossible car on change de case. Ensuite on vérifie que les cases de coordonnées (xtab+i ;ytab-i) (donc de la diagonale haute droite) sont pleines, si la pièce faisant obstacle est ennemie alors casepleine=0 (sinon elle est alliée alors la valeur est 1).
- La diagonale haute gauche : même concept sauf que cette fois les cases du tableau à vérifier son les cases de coordonnées (xtab-i ; ytab-i) car on va vers la gauche de l'axe x
- La diagonale basse droite : cette fois les valeurs de i vont aller de c2 à 1, tout simplement parce qu'ici y2tab>ytab, et les cases vérifiées sont les cases de coordonnées (xtab+i ;ytab+i). Sinon tout est identique.
- La diagonale basse gauche : même chose que pour la diagonale basse droite mais avec les cases vérifiées de coordonnées (xtab-i ;ytab+i).

Une condition que l'on vérifie également est la valeur de c1, car si celle-ci est 0 alors cela signifie que la case d'arrivée sera la même que celle de départ ce qui est impossible.
Enfin, comme pour la tour, le déplacement est conforme lorsque la case visée est vide (casepleine=0).

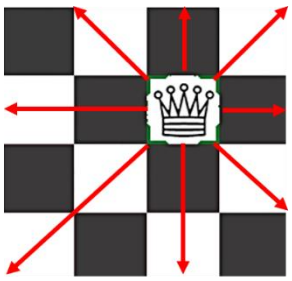


Figure 7 : Déplacement de la reine

Pour terminer cette procédure, le déplacement de la tour est en réalité assez simple puisqu'il suffit de combiner les déplacements du fou et de la tour. A noter tout de même qu'il est nécessaire d'introduire deux autres variables « casepleine2 » et « constante2 », qui ont la même fonction que leurs homologues puisque l'on doit vérifier les déplacements en ligne droite et ceux en diagonale.

B. Echek

Passons à présent à la procédure « Echek », qui comme son nom l'indique cherche à dire si le roi, qu'il soit noir ou blanc (cette distinction est faite avec des tests disjoints, un pour le joueur blanc et un pour le noir). Pour cela on fait l'hypothèse que le roi n'est pas en échec, et on cherche à montrer l'inverse. Cependant, avant de pouvoir tester toutes les conditions où le roi pourrait se retrouver en échec, il faut d'abord savoir où ce dernier se trouve sur le plateau. A cet égard, on parcourt le plateau à l'aide d'une double boucle parcourant les 64 cases du damier, on récupère ainsi les coordonnées pour chacun des rois.
En réalité, comme pour la procédure expliquée précédemment, il suffit d'expliquer uniquement le fonctionnement de la vérification de la situation d'échec pour une des deux couleurs. En effet, par analogie tous les raisonnements suivis pour le roi blanc seront les mêmes que pour le roi noir, avec seulement des adaptations de valeurs, et des adaptations au niveau des pièces contenues dans les cases testées – les pièces ennemies sont différentes en fonction de la couleur des pièces du joueur en cours. De ce fait, à la place d'utiliser les notations exactes du code (xroib, xroin, yroib et yroin), on utilisera la notation (xroi, yroi) puisque la couleur n'est pas importante)

Dans tous les cas, on peut séparer les vérifications en 4 types de déplacements : le cas du cavalier, celui du pion, les déplacements en ligne droite et ceux en diagonales.

Tout d'abord, pour un cavalier ennemi on fait cette vérification à l'aide d'une boucle allant de -2 à 2, comme le cavalier se déplace en L, il peut s'agir d'un L à la gauche du roi ou bien à sa droite, d'où les valeurs -2 et 2.
Si le compteur i_echec est autre que 0 alors on peut remarquer 2 cas distincts :

- Soit le cavalier ennemi est décalé d'un L dans le sens des y descendants (donc vers le haut du plateau),on fait le postulat que le roi est en échec puis on cherche à réfuter ce postulat pour le démontrer. Pour annuler ce postulat on regarde si xroi+i_echec ou yroi-abs(3-abs(i_echec)), correspondant aux coordonnées de la case du cavalier, sont telles que ce cavalier devrait être hors des limites du plateau. A ce moment l'échec supposé est réfuté puisqu'il s'agirait en quelque sorte d'un « échec imaginaire » ce qui est impossible ;
- Soit le cavalier est décalé d'un L avec les y montants (donc vers le bas du plateau), et par analogie on vérifiera ici que les coordonnées (xroi+i_echec ; yroi+abs(3-abs(i_echec)) ne sont pas telles que la case est hors limites.

A noter que lorsque i_echec vaut 0 l'échec est considéré comme faux.

Ensuite, du point de vue des pions adverses, on fait l'hypothèse que l'échec n'est jamais vérifié cette fois-ci, puis on cherche à montrer l'inverse. Il y a deux conditions possibles pour pouvoir affirmer que le roi est en échec (voir la figure ci-après pour le cas de l'échec du roi blanc) :

```
//pion ennemi
if ((tab[xroib-1,yroib-1]='pionn')and(xroib-1>=1)and(yroib-1>=1)) or ((tab[xroib+1,yroib-1]='pionn')and(xroib+1<=8)and(yroib-1>=1)) then
begin echec:=true end;
```

Figure 8 : Vérification de mise en échec du roi blanc par un pion noir

Comme on le voit, pour chacune des conditions on regarde si la case de coordonnées (xroi+/-1 ;yroi+/-1) contient bien un pion ennemi. Ce différence de signe +/- s'explique par le fait que ce dernier va dépendre du joueur que l'on est en train de considérer : dans ce cas on considère le joueur 1 qui jouent les pièces blanches, donc les pions adverses sont les noirs et ceux-ci ne peuvent que descendre le long de l'axe y, concernant les coordonnées en x ça va dépendre de si le pion arrive à la gauche ou à la droite du roi. Dans tous les cas on s'assure que le fait d'enlever ou d'ajouter 1 ne fait pas « sortir » la case testée du plateau.

A présent, pour les déplacements en ligne droite, la vérification est un peu plus complexe, mais une fois de plus, une fois que le principe est détaillé pour les déplacements verticaux par exemple, on peut en déduire la méthode de vérification pour les déplacements horizontaux. Pour commencer on pose une variable wow initialisée à 0, puis comme pour la tour dans la procédure « Dpt Valide », on distingue 2 cas.
Premièrement, les mouvements verticaux on regarde avant toute chose la valeur de yroi, si elle vaut 1 (dans le cas de la verticale haute) ou 8 (dans le cas de la verticale basse), on attribuera à wow la valeur -1. Par la suite, on va utiliser une boucle indéterministe (repeat…until) où l'on commence par poser wow =wow+1 pour ensuite pouvoir utiliser une autre variable wow2=wow dans une condition se trouvant à l'intérieur de la boucle. Le fait de poser wow2 va nous permettre de nous servir de la valeur de « début de boucle » de wow, donc avant de rentrer dans la condition (if…then), cela va imposer que wow2 ne puisse jamais prendre d'autres valeurs que 0 ou 1.
De là, on vérifie que la case (xroi ;yroi-wow) – dans le cas de la verticale haute mais avec un signe + au lieu de - pour la basse – est occupée par la reine, une tour ou le roi adverse ; si c'est le cas on affectera la valeur 7 à wow pour nous permettre, lorsque la condition est vraie, de sortir de la boucle. Bien sûr il est essentiel que wow2=1 pour que la condition soit vraie, puisque sinon on ne vérifierait pas la bonne case.
Lorsque cette condition est vraie, le roi est en échec, sinon il ne l'est pas.

Cela étant dit, même si la condition était fausse, on serait toujours capable de sortir de la boucle car la 2ème condition possible pour sortir de cette dernière est que yroi-wow2=1, ce qui se rapporte à notre première vérification. En effet, la seule possibilité ici est que yroi=1 et wow2=0, et donc que wow=-1. Le fait de donner ces deux possibilités de sortie de boucle, nous assure que l'itération sera bien faite qu'une seule fois, puisque finalement le résultat au bout de cette seule itération nous permet de conclure sur l'échec ou non du roi.
Deuxièmement, pour les déplacements horizontaux, on suit exactement le même raisonnement, mais avec de nouveau une différenciation entre les déplacements à gauche et ceux à droite. Ici, ce seront les coordonnées xroi qui seront modifiés et vérifiés puisque l'on se déplace sur une droite y=yroi.

Pour finir, au niveau des déplacements diagonaux, exactement le même principe que lorsque l'on a testé la validité des déplacements du fou dans la procédure « Dpt Valide » : on découpe les mouvements avec la diagonale haute droite, gauche, diagonale basse droite, gauche.
Pour les déplacements diagonaux, on fait comme pour les déplacements du fou dans la procédure « Dpt Valide », on découpe les mouvements : diagonale haute droite, gauche, diagonale basse droite, gauche. Non seulement cela, mais la manière de vérifier l'échec suit la même logique que pour les déplacements en ligne droite.
Cette fois les 2 coordonnées (xroi ;yroi) sont vérifiées et modifiées par wow et wow2 puisque l'on se déplace en diagonale. De plus, dans la condition à l'intérieur de la boucle, on vérifie cette fois-ci que la case testée contient ou non la reine, le roi ou un fou ennemi. Les conditions pour pouvoir quitter la boucle sont différentes de celles pour les déplacements en ligne droite, ici il faudra wow=7, ou bien que les valeurs des différences ou additions de xroi et wow2 et de yroi et wow2. Par exemple pour la diagonale haute droite les conditions de sortie sont :

```
    end;
until (wow=7)or(xroib+wow2=8)or(yroib-wow2=1);
//fin diagonale haute droite
```

Figure 9 : Conditions de sortie de boucle pour la diagonale haute droite lors de la vérification de l'échec du roi blanc

C. EchekMat

La dernière procédure détaillée dans ce rapport concerne la procédure s'occupant de vérifier si un roi est en échec et mat. Cette procédure est essentielle au bon fonctionnement du jeu puisque sans elle aucun vainqueur ne peut jamais être déclaré – la partie pourrait se terminer en pat mais le résultat final reste le même, il n'y aura pas de vainqueur.

Pour cette procédure, il nous suffit de détailler la vérification de l'échec et mat pour une seule pièce quelle qu'elle soit, car tout le raisonnement est le même pour toutes les pièces, la seule différence sera le joueur que l'on considère en train de jouer (le 1 ou le 2), puisque pour chacun des joueurs on testera toutes les pièces lui appartenant.
Au départ, pour toutes les vérifications, on part du principe que l'échec et mat est vrai, puis selon la même démarche que pour certaines vérifications de la procédure « Echek », on cherche toutes les possibilités pouvant réfuter cette hypothèse puis on les teste.

En premier lieu, on parcourt tout le plateau à l'aide d'une double boucle – donc les 64 cases – pour connaître l'emplacement des différentes pièces puisqu'on les traite au cas par cas, les coordonnées récupérées seront alors (xtab ;ytab). Ensuite, le cas par cas commence.
Prenons une pièce quelconque de coordonnées (xtab ;ytab). De là, on va poser que cette pièce se trouve dans une case initiale que l'on appelle echekmat_p1, et on va tenter de déplacer cette dernière dans toutes les cases du plateau. Pour cela, on utilise une deuxième double boucle, avec les coordonnées (x2tab ;y2tab). On attribue ces coordonnées à une nouvelle pièce que l'on nomme echekmat_p2, puis on vérifie que le déplacement de la pièce echekmat_p1 à echekmat_p2 est bien

valide grâce à la procédure « Dpt Valide » – car finalement echekmat_p2 représente la pièce dans la case d'arrivée de echekmat_p1, donc on peut la considérer comme l'état final de echekmat_p1. Si le déplacement est validé, on déplace la pièce dans la case (x2tab ;y2tab) et on regarde si cette pièce met le roi adverse à l'aide de « Echek ».
Lorsqu'il n'y a pas d'échec alors le booléen echec devient faux, il n'y a donc pas de situation d'échec et mat. Dans tous les cas une fois toutes les vérifications faites, on replace la pièce testée dans sa case d'origine, de coordonnées (xtab ;ytab). On fait de même pour toutes les pièces.

Ce déroulement peut être résumé en disant que l'on déplacera chaque pièce, en fonction du joueur considéré toujours, pour voir si elle met en échec le roi adverse, si c'est le cas alors l'échec et mat est vérifié. Puis, on remettra ces mêmes pièces dans leur position d'origine respective.

```
procedure echekmat(a : integer; var echecmat : boolean);
var xtab,ytab,x2tab,y2tab : integer;
    echekmat_p1,echekmat_p2 : string;
    vf,echec,roque : boolean;
begin
    echecmat:=true;
    if a=1 then
    begin
        for xtab:=1 to 8 do
            for ytab:=1 to 8 do
            begin
                if tab[xtab,ytab]='pionb' then
                    begin
                    echekmat_p1:=tab[xtab,ytab];
                    for x2tab:=1 to 8 do
                        for y2tab:=1 to 8 do
                        begin
                            echekmat_p2:=tab[x2tab,y2tab];
                            deplacementvalide(xtab,x2tab,ytab,y2tab,a,echekmat_p1,vf,roque);
                            if vf=true then
                                begin
                                tab[xtab,ytab]:='vide';
                                tab[x2tab,y2tab]:=echekmat_p1;
                                echek(a,echec);
                                if echec=false then
                                    begin echecmat:=false end;
                                tab[xtab,ytab]:=echekmat_p1;
                                tab[x2tab,y2tab]:=echekmat_p2;
                                end;
                        end;
                    end;
```

Figure 10 : Exemple utilisé pour la description de « EchekMat »

IV. Les problèmes et difficultés rencontrés

Tout au long de l'écriture du code, certains obstacles ont été rencontrés et donc réglés pour permettre le bon avancement du projet. Ces problèmes ont été aussi bien des problèmes lors de l'écriture elle-même sur lesquels il a fallu réfléchir à une solution, mais également des problèmes d'ordre plutôt matériel qui ont dû être contournés pour pouvoir avancer.

Dans un premier temps, du point de vue de l'écriture du code, un des problèmes fût rencontré lors de la création du damier. Au départ, l'idée avait été de faire s'afficher les cases une par une, mais cette manière de faire était inutile puisque l'on peut remarquer une parité de la somme des coordonnées des cases. Ainsi, le programme est moins long et plus lisible.
Ensuite, un autre problème était lié à l'utilisation du fichier « save ». En effet, initialement les pièces du plateau était de type « pièce », or il était impossible de créer un fichier de type pièce (File of 'pièce'). De ce fait, nous avons décidé d'attribuer le type « String » aux pièces, puisque cela revenait alors à créer un fichier de type string (File of 'string') ce qui est faisable. En plus de cela, des soucis avec le fichier text de la sauvegarde étaient également observables, puisque celui-ci se sauvegardait mal et lorsqu'il était chargé (Load), une mauvaise sauvegarde apparaissait. Ce problème a été résolu en forçant le programme à prendre le temps de faire la sauvegarde, ce qui n'était avant pas toujours le cas et donc le programme n'avait pas toujours le temps d'écrire toutes les lignes à sauvegarder dans le fichier.
Le dernier type de soucis d'écriture rencontrés étaient quant à eux liés à la SDL, avec notamment sa mise en place. Effectivement, le programme devait être seulement affiché sur le terminal, mais dans un souci d'esthétisme nous avons décidé de faire un programme en SDL (Simple DirectMedia Layer). Il a donc fallu pour cela apprendre à utiliser la SDL, et créer toutes les images des pièces, des cases et des curseurs. Un autre problème avec la SDL était le fait que pour l'affichage du texte, il est théoriquement possible d'utiliser « sdl_rendertext », malheureusement nous n'avons pas compris comment s'en servir. Du coup, les images sont au final créées avec les textes dedans, ce n'est pas l'idéal, mais c'était plus simple pour l'écriture.
Par ailleurs, une difficulté d'écriture fût la procédure « Timer », comme celle-ci utilisait le temps actuel de l'ordinateur grâce à la fonction « gettime », il était essentiel que lorsque les joueurs décidaient de quitter une partie avant la fin (de la mettre en pause en quelque sorte) que le Timer ne décompte pas les minutes pendant la pause. Par exemple, si les joueurs quittaient la partie pendant 5 minutes, il a fallu faire attention au fait que le Timer n'enlève pas ces 5 minutes au Timer de chacun des joueurs.

Dans un second temps, d'un point de vue logistique cette fois-ci, le partage du code entre les différentes personnes du groupe était compliqué, puisqu'il fallait systématiquement s'envoyer les dernières versions du programme – à travers le groupe Messenger créé à cet effet – si l'on voulait travailler dessus de notre côté. Autrement il aurait fallu rechercher et mettre à jour les deux versions à chaque fois (celle avec SDL et celle sans).

Le fait d'avoir rencontré des problèmes lors de la conception du programme nous a permis de développer nos capacités à résoudre des obstacles à l'avancée de notre travail. Ces capacités qui nous seront bien évidemment nécessaires non seulement pour les futurs travaux de groupe que nous devrons réaliser tout au long de nos études, mais aussi plus tard lorsque nous serons dans le monde du travail sur des projets.

Conclusion

Pour conclure, nous pouvons dire que compte tenu des connaissances et des outils qui nous ont été fournis en première année, nous n'étions au début pas persuadés d'arriver à rendre un projet aussi abouti. Effectivement, toutes les idées et fonctionnalités que nous avions listé lors de la phase de conception globale du projet ont, non seulement été produites, mais de nouvelles options de jeu ont été rajoutées au fur et à mesure de l'avancement du projet. La persévérance dont nous avons dû faire preuve pour passer outre les problèmes et difficultés rencontrés a permis de développer nos capacités à réfléchir et chercher des solutions, mais nous a surtout permis de réaliser le programme que nous souhaitions. Un des points le plus important est d'avoir réussi à utiliser un fichier pour pouvoir stocker tous les paramètres d'une partie qui a été « mise en pause », dans le but de permettre aux joueurs de reprendre leur partie dans les mêmes conditions que lorsqu'ils l'avaient quitté et de pouvoir terminer cette partie.

Pour l'organisation du travail, malgré quelques problèmes pour se partager les différentes versions du code au fur et à mesure de son écrite et de ses mises à jour, une bonne communication a permis au groupe de ne pas avoir de quiproquos, ou de désaccords, ainsi le projet a pu avancer efficacement. Par ailleurs, aux vues des différences de niveau entre les différentes personnes du groupe, nous avons réussi à nous répartir le travail pour permettre à tous les membres de n'avoir pas uniquement une bonne compréhension du programme, mais aussi et surtout d'avoir des tâches cohérentes à réaliser prenant en compte les facilités et compétences de chacun.

En dernière analyse, nous pouvons ajouter que des améliorations, ou plutôt approfondissements, que nous pourrions envisager l'utilisation de la souris pour sélectionner les pièces, malheureusement ceci n'est pas faisable si nous restons en Pascal. Enfin, pour pousser encore plus loin ce projet, la prochaine étape serait sans doute l'Intelligence Artificielle pour qu'il ne soit pas nécessaire d'avoir deux joueurs humains pour pouvoir jouer une partie d'échecs, mais bien sûr il nous faudrait beaucoup plus de connaissances et de compétences que celles que nous avons actuellement.

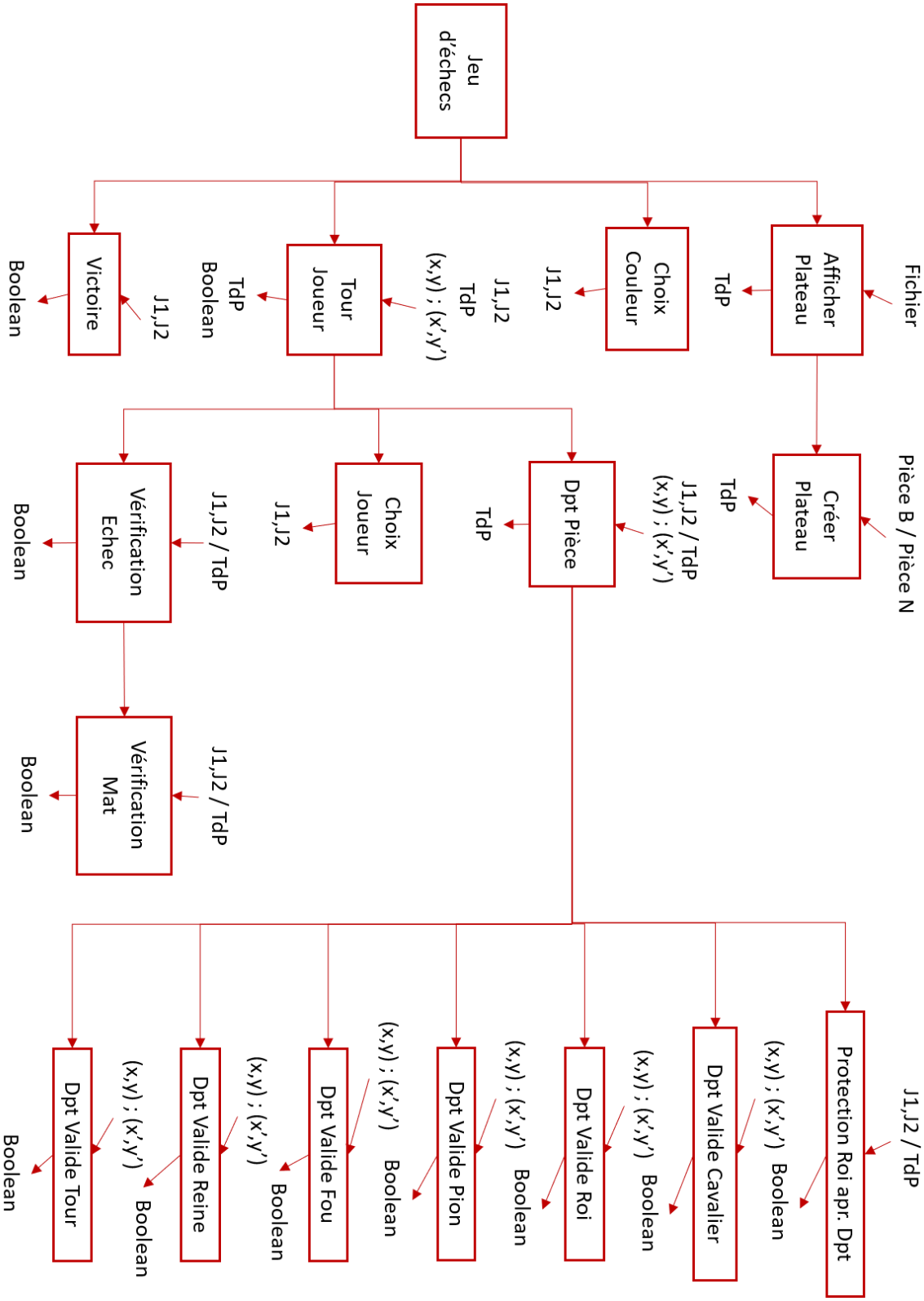
Annexe 1 : Cahier des charges (version 2)

La mise à jour du cahier des charges a été essentiellement faite du point de vue des fonctionnalités du programme.
L'objectif de ce programme quant à lui reste globalement inchangé, les seuls ajouts sont d'une part certains coups spéciaux ayant été rajoutés, et d'autre part la possibilité pour les joueurs de sauvegarder une partie pour pouvoir la reprendre plus tard.

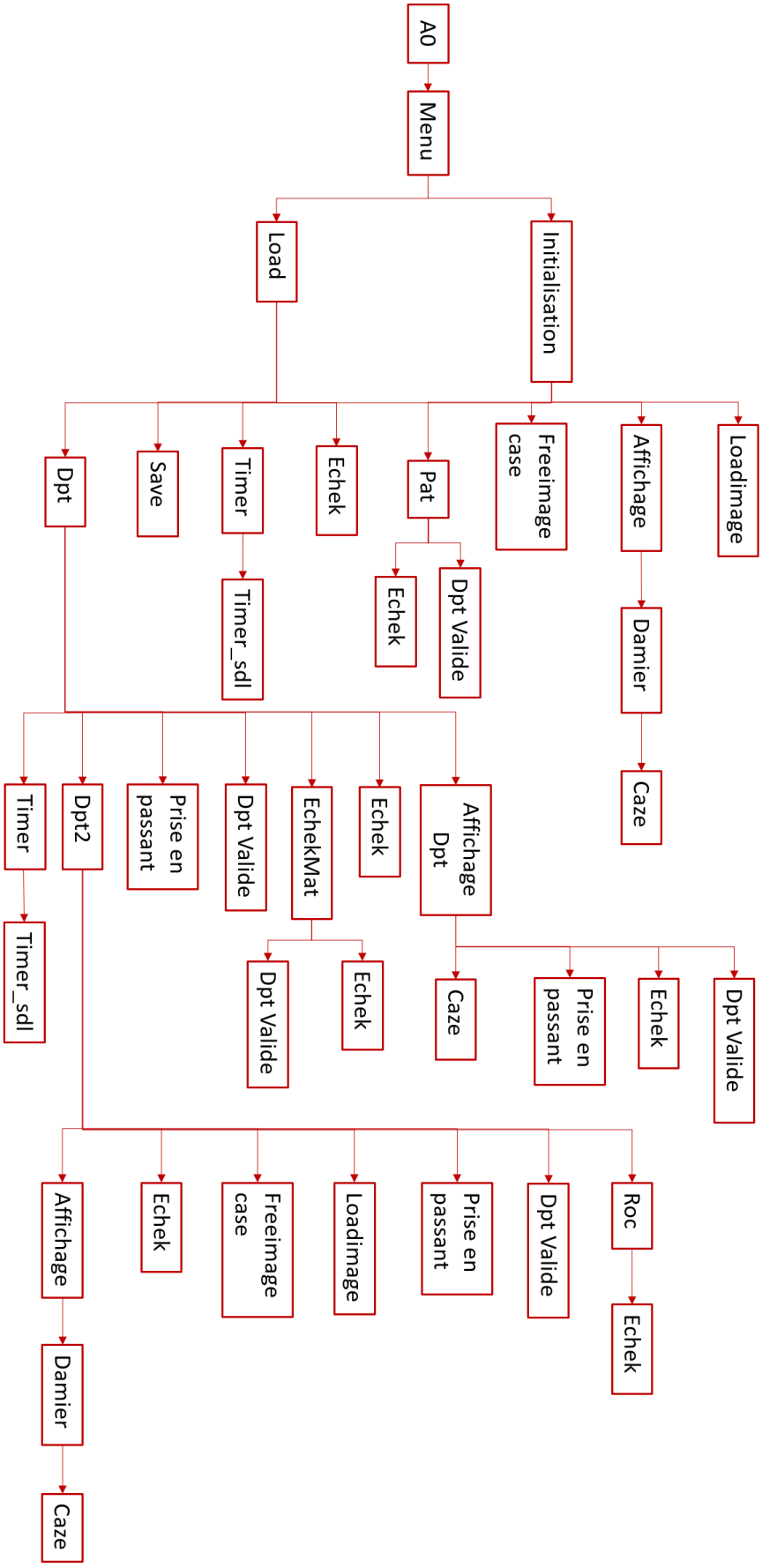
Voici ci-après la liste des fonctionnalités finales du programme :

- Pouvoir choisir soit de commencer une nouvelle partie, soit de reprendre une partie sauvegardée ;
- Pouvoir sauvegarder une partie ;
- Afficher un plateau de jeu ;
- Actualiser le plateau à chaque déplacement (et donc réafficher le « bon » plateau) ;
- Afficher les différentes pièces ;
- Pouvoir déplacer les pièces à l'aide du clavier ;
- Vérifier qu'un déplacement est valide – c'est-à-dire qu'il soit en accord avec les règles du jeu d'échecs ;
- Permettre le coup du « Roque » ;
- Permettre « la Prise en passant » ;
- Afficher un Timer ;
- Vérifier si un joueur est en échec ;
- Vérifier si un joueur est échec et mat ;
- Vérifier si le jeu est « pat ».

Annexe 2 : Analyse descendante (version 1)



Annexe 3 : Analyse descendante (version 2)



Bibliographie

Sources pour la description du jeu d'échecs classiques, et des différents coups spéciaux :

https://fr.wikipedia.org/wiki/Règles_du_jeu_d'échecs
http://www.creachess.com/cours/le-roque.php
https://fr.wikipedia.org/wiki/En_passant_(échecs)