Dossier de conception préliminaire

MAHÉ François
POMERET-COQUOT Pierre

francois.mahe@univ-tlse3.fr
pierre.pomeret@univ-tlse.fr

Vendredi 24 mars 2017

Dossier de conception préliminaire

But du document

Est présenté ici le fonctionnement général du logiciel en développement, par la description de ses différents modules.

Pour chacun des modules : description, relations avec les autres modules, calendrier de réalisation et plan de tests / validation.

Vous trouverez en annexe des diagrammes et les TAD des structures de données utilisées en C.

Présentation générale

Le logiciel se comporte d'une interface utilisateur réalisée en Java, laquelle utilise des bibliothèques natives écrites en C, et utilisables grâces à JNI.

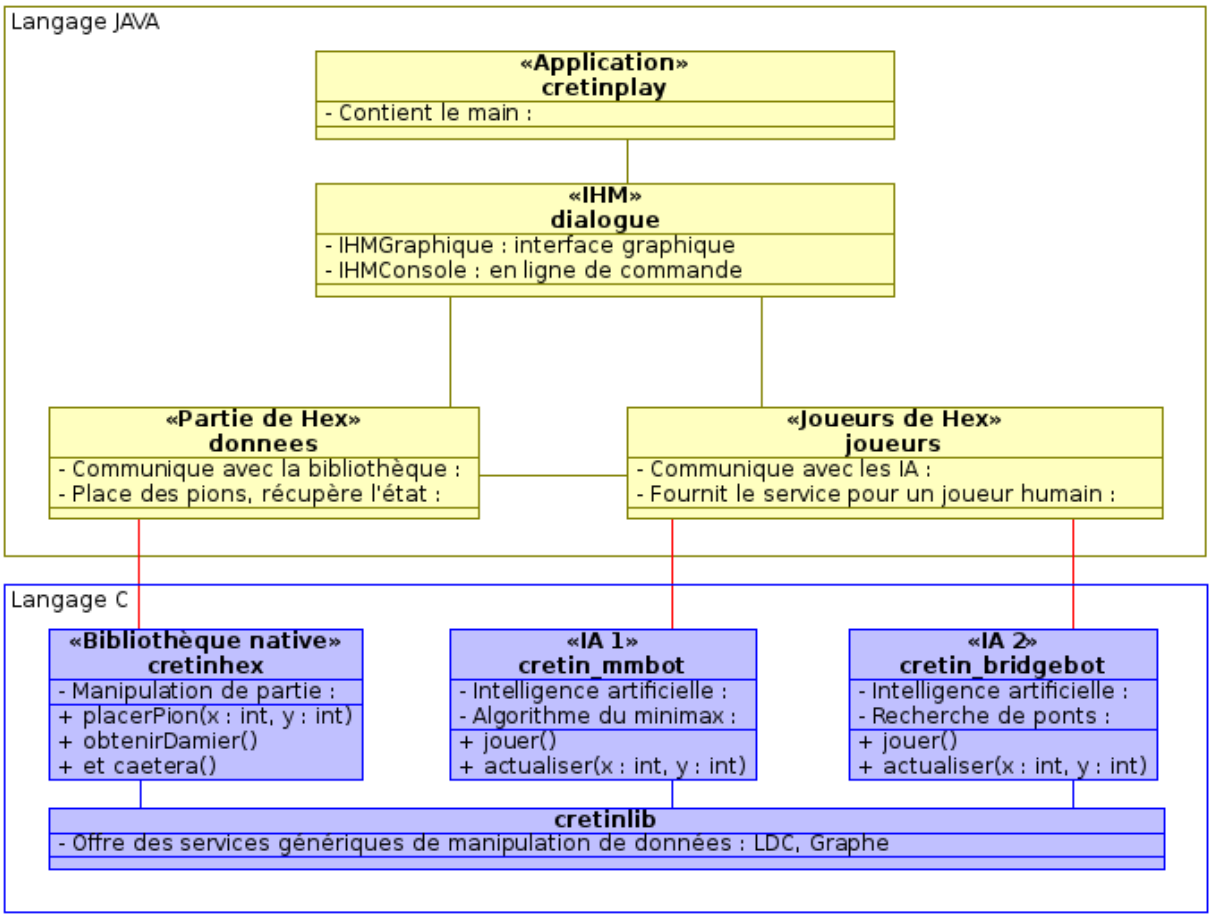


Schéma (très) simplifié de l'organisation du programme

DESCRIPTION DES DIFFÉRENTS MODULES

1, LANGAGE JAVA

L'application Java est constituée de 5 modules :

- Classe *main* interprétant les éventuels paramètres passés en ligne de commande
- Interface en ligne de commande
- Interface graphique
- Manipulation des données de jeu
- Utilisateurs (joueurs humains en non humains)

Et d'une méthode *main* dans la classe *Application*.

Cela est détaillé dans le diagramme de classe UML en annexe.

Un exemple de diagramme de séquence illustre le fonctionnement de ces modules.

Nom	**Application**
Rôle	Méthode main, interprétation des paramètres passés en ligne de commande
Type de données	N/A
Dépendances	Tous les modules suivants
Liste des fonctionnalités fournies	Instancie l'interface graphique ou en ligne de commande, et appelle sa commande *play()*. Interprète les paramètres passés en ligne de commande

Nom	**IHMConsole**
Rôle	Interface utilisateur en ligne de commande
Type de données	Page : affichage des informations Menu : Interaction avec l'utilisateur
Dépendances	Module Partie
Liste des fonctionnalités fournies	Interface utilisateur. Grâce aux menus l'utilisateur peut naviguer au travers de différentes pages pour créer / charger / etc. des parties. De cette façon il atteindra la page de jeu qui lui permettra de jouer (placer des pions), grâce au module Partie décrit ci-dessous.

Nom	**IHMGraphique**
Rôle	Interface utilisateur graphique
Type de données	Cadre : le contenu de la fenêtre
Dépendances	Module Partie
Liste des fonctionnalités fournies	Interface utilisateur. Les fonctionnalités fournies seront les mêmes que celles du module IHMConsole, mais en mode graphique

Nom	**Partie**
Rôle	Dialogue avec la bibliothèque native cretinhex
Type de données	Partie : fonctionnalités pour java PartieJNI : dialogue avec la bibliothèque native cretinhex
Dépendances	Bibliothèque native *cretinhex*
Liste des fonctionnalités fournies	Manipulation de partie (placement de pions, sauvegarde / restauration de parties, consultation du damier.

Nom	**Joueur**
Rôle	Dialogue avec les bibliothèques des AI
Type de données	Utilisateur et classes héritées (Humain, AI1, AI2)
Dépendances	Bibliothèque native *cretinhex_mmbot* et *cretinhex_bridgebot*
Liste des fonctionnalités fournies	Implémente deux fonctions permettant de récupérer le coup joué par une IA (ou le coup (-1,-1) pour un humain), et d'informer cette IA des autres coups joués.

2. LANGAGE C

La partie C est découpée en quatre modules :

- Structures de données génériques (cretinlib)
- Structure de données pour la manipulation de partie
- Intelligence artificielle implémentant l'algorithme du minimax
- Intelligence artificielle implémentant une stratégie de recherche de ponts.

Cela est détaillé en annexe dans le pseudo-diagramme de classes.

Nom	**cretinlib**
Rôle	Structures de données génériques
Type de données	LDC, Graphe
Dépendances	N/A
Liste des fonctionnalités fournies	Structures de données et fonctions associées *(cf TAD en annexe)*

Nom	**cretinhex**
Rôle	Manipulation de parties de Hex + interface JNI
Type de données	Partie, Damier, GrapheHex.
Dépendances	cretinlib
Liste des fonctionnalités fournies	Instanciation de parties, placement de pions, consultation du plateau et de l'historique, et sauvegarde. *(cf. TAD en annexe)*

Nom	**cretinhex_mmbot**
Rôle	AI1 : décision avec minimax + interface JNI
Type de données	Arbre Minimax
Dépendances	cretinlib, cretinhex
Liste des fonctionnalités fournies	Implémente les fonctions *jouer()* et *actualiser() (cf. TAD en annexe)*

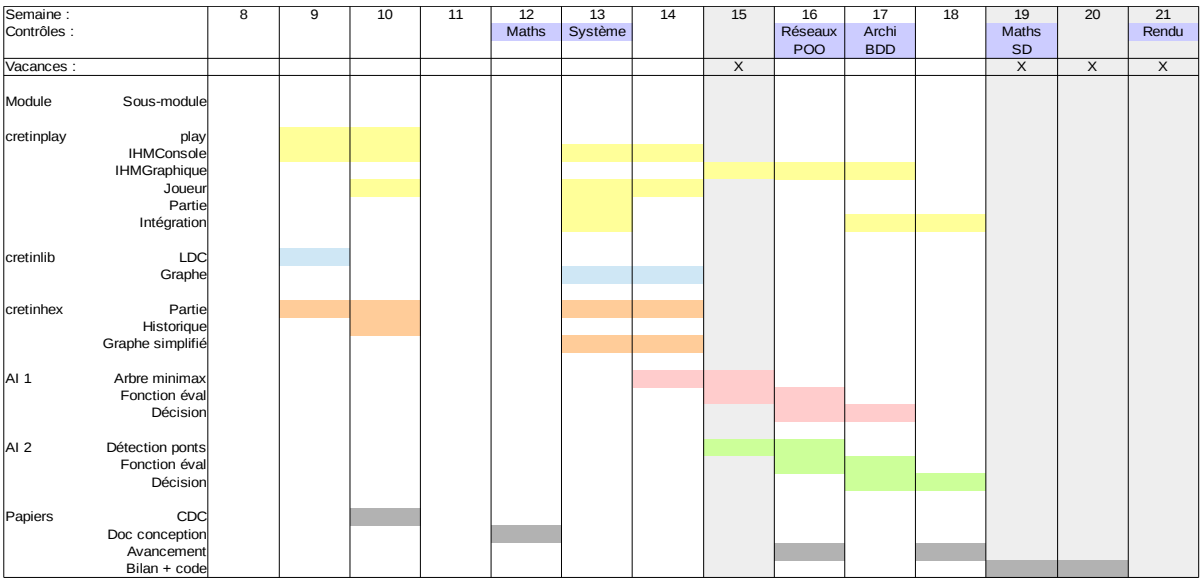
Nom	**cretinhex_bridgebot**
Rôle	AI 2 : détection de ponts + interface JNI
Type de données	N/A
Dépendances	cretinlib, cretinhex (Damier, GrapheHex)
Liste des fonctionnalités fournies	Instanciation de parties, placement de pions, consultation du plateau et de l'historique, et sauvegarde. *(cf. TAD en annexe)*

RÉPARTITION DES TÂCHES ENTRE CHAQUE MEMBRE DU GROUPE

Tâche(s)	Membres
JAVA : Module ***cretinplay***	François
JAVA : Module ***IHMConsole***	François
JAVA : Module ***IHMGraphique***	François et Pierre
JAVA : Module ***Partie***	Pierre
JAVA : Module ***Joueurs***	François et Pierre
C : Module ***cretinlib***	Pierre
C : module ***cretinhex***	Pierre
C : Module ***cretinhex_mmbot***	François
C : Module ***cretinhex_bridgebot***	Pierre

Sans jamais oublier que le travail d'équipe se fait en équipe :-)

CALENDRIER DE RÉALISATION DES TACHES



Les deux dernières semaines avant le rendu du projet constituent une marge de 10 % pour palier aux imprévus, ou peaufiner les finitions.

PLAN DE TESTS

STRUCTURES DE DONNÉES RÉALISÉES EN C

Les tests de la partie C sont tous automatisés. De courts programmes permettent de réaliser les opérations élémentaires décrites ci-dessous, sur de grandes listes de données (nombres aléatoires, sauvegardes de parties).

Ces programmes sont appelés par un script *shell* et leur résultat est comparé avec le résultat attendu (i.e. calculé un autre programme, idéalement développé selon un autre algorithme)

Tests automatisés :

- LDC, Graphe : insertion, suppression, restitution de collections d'entiers
- Partie : chargement, détection du vainqueur ou chargement, placement de pion et sauvegarde. Essais de coups impossibles
- AI1 : création de l'arbre Minimax, fonction d'évaluation. Recherche du meilleur résultat.
- AI2 : liste des ponts, Recherche du meilleur résultat.

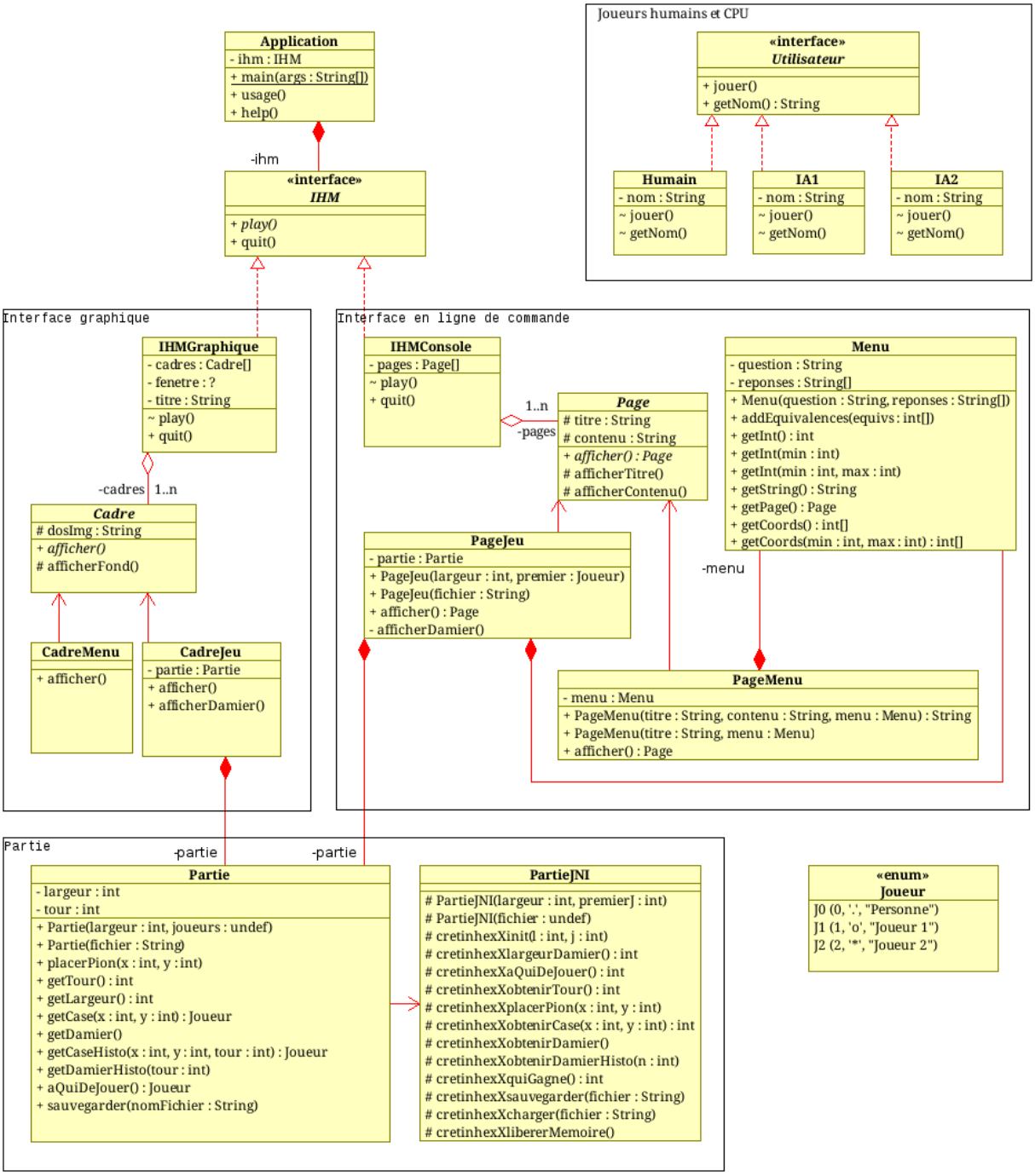
CLASSES DE LA PARTIE JAVA

La recherche de bogues sur l'interface développée en Java se fera de plusieurs façons complémentaires :

- L'intégration des IA et de la bibliothèque *cretinhex* est validée en lançant un grand nombre de parties IA contre IA (une IA jouant au hasard sera développée à cette fin).
- Les interfaces graphiques et en ligne de commandes sont / seront construites d'après des principes simples d'IHM (i.e. élaboration du schéma du dialogue, tests d'inspections et tests avec des utilisateurs pour la version graphique), afin d'assurer une parfaite utilisabilité, et de démasquer les éventuels bogues.

ANNEXES

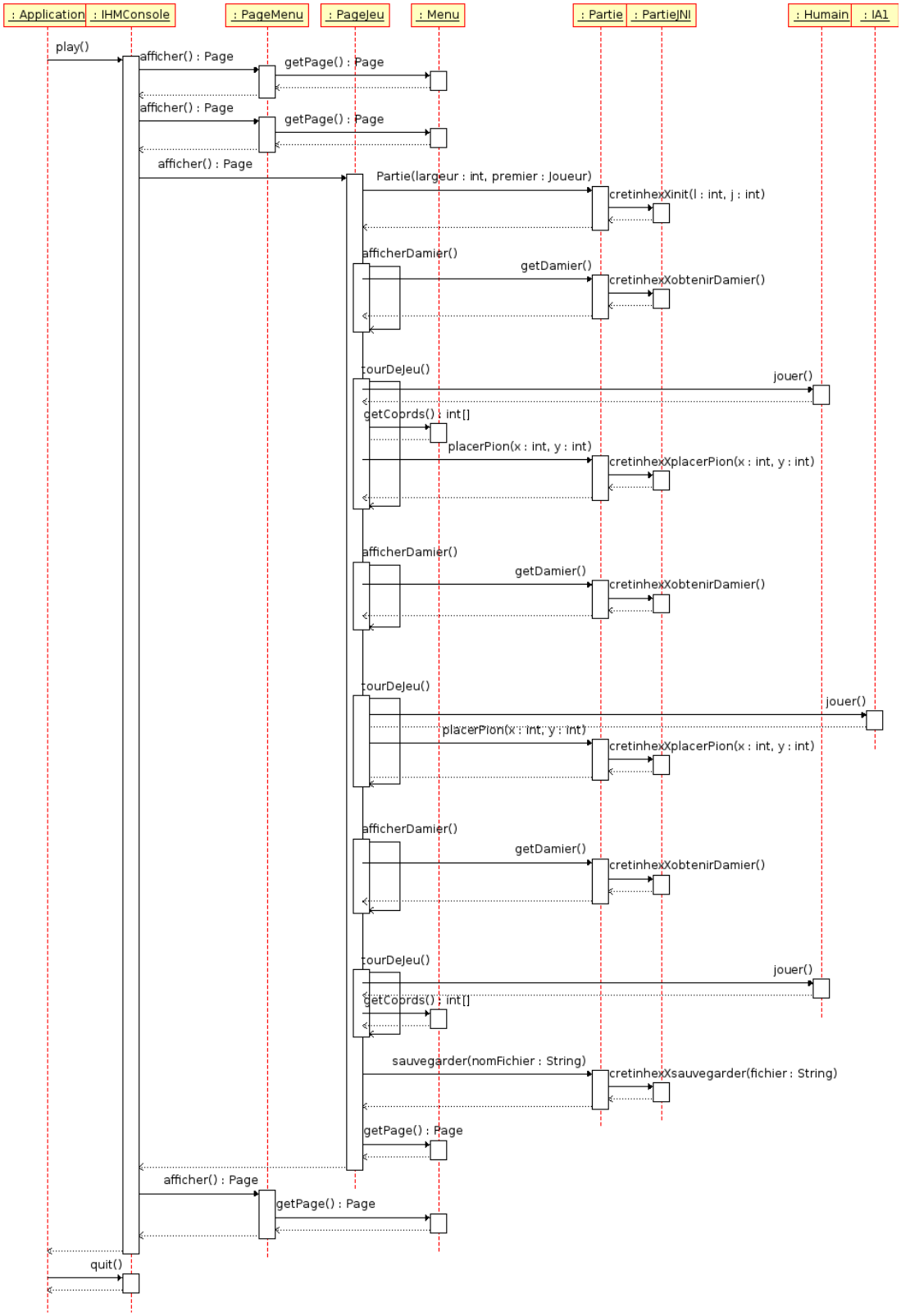
JAVA : DIAGRAMME DE CLASSES



Note : Le module « Interface Graphique » n'est pas encore réellement défini, il devra respecter l'interface IHM. Par ailleurs, nous hésitons encore à rattacher les deux joueurs (Utilisateur) à la classe Application ou à la classe Partie.

JAVA : DIAGRAMME DE SÉQUENCE

Un exemple de fonctionnement de l'application JAVA : deux coups sont joués.



L'IHM utilisée est IHMConsole. Seul un joueur humain peut quitter.

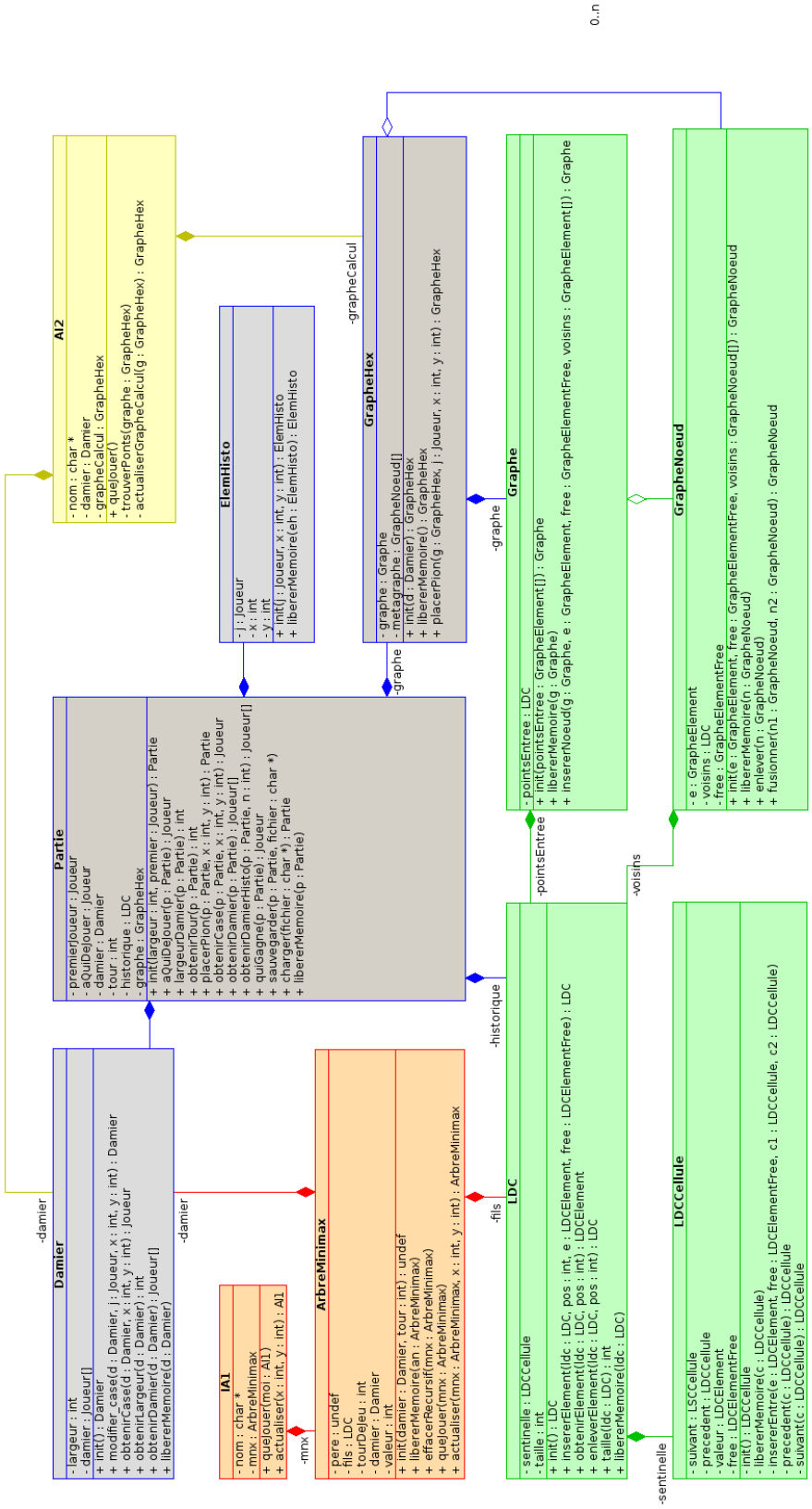


Diagramme illustrant les structures de données et leurs dépendances, dans la partie C

C : TAD DES STRUCTURES DE DONNÉES

LDC

Sorte :

- LDC

Utilise :

- LDCElement (void *)
- LDCElementFree (fonction supprimant l'élément)
- int

Constructeurs :

- init : __ → LDC
- insererElement : LDC x LDCElement x LDCElementFree x int → LDC

Opérateurs :

- obtenirElement : LDC x int → LDCElement
- enleverElement : LDC, int → LDC
- taille : LDC → int
- libererMemoire : LDC → __

Préconditions :

- insererElement(ldc, e, free, i) → taille(ldc) ≥ i
- obtenirElement(ldc, i) → 0 < i ≤ taille(ldc)
- enleverElement(ldc, i) → 0 < i < taille(ldc)

Axiomes :

- obtenirElement(insererElement(ldc, e, free, i), j) =
  - e si i = j
  - obtenirElement(ldc, j) si i > j
  - obtenirElement(ldc, j + i) si i < j
- enleverElement(insererElement(ldc, e, free, i), j) =
  - ldc si i = j
  - insererElement(enleverElement(ldc, j), e, free, i) si i > j
  - insererElement(enleverElement(ldc, j-1), e, free, i -1) si i < j
- taille(init()) = 0
- taille(insererElement(ldc, e, free, i) = taille(ldc) + 1

GRAPHENOEUD

Sorte :

- GrapheNoeud

Utilise :

- GrapheElement (void *)
- GrapheElementFree (fonction supprimant l'élément)

Constructeur :

- init : GrapheElement, GrapheElementFree → GrapheNoeud

Opérateurs :

- obtenirElement : GrapheNoeud → GrapheElement
- obtenirVoisins : GrapheNoeud → GrapheNoeud[]
- fusionner : GrapheNoeud x GrapheNoeud → GrapheNoeud
- libererMemoire : GrapheNoeud → __

GRAPHE

Sorte :

- Graphe

Utilise :

- GrapheNoeud

Constructeurs :

- init : pointsEntree[] → Graphe
- insererNoeud : GrapheNoeud x GrapheNoeud[] → Graphe

Opérateurs :

- libererMemoire : Graphe → __

Note : Les TAD du Graphe et de ses nœuds ne sont probablement pas complets, c'est d'ailleurs l'un des travaux prévus pour la semaine prochaine.

DAMIER

Sorte :

- Damier

Utilise :

- Joueur (enum J0, J1, J2)
- int

Constructeurs :

- init : int → Damier
- modifierCase : Damier x Joueur x int x int → Damier

Opérateurs

- obtenirLargeur : Damier → int
- obtenirCase : Damier x int x in → Joueur
- obtenirDamier : Damier → Joueur[]
- libererMemoire : Damier → __

Préconditions :

- modifierCase(d, j, x, y) → 0 ≤ x, y < obtenirLargeur(d)
- obtenirCase(d, x, y) → 0 ≤ x, y < obtenirLargeur(d)

Axiomes :

- obtenirLargeur(init(l)) = l
- obtenirLargeur(modifierCase(d, j, x, y)) = obtenirLargeur(d)
- obtenirCase(init(l), x, y) = J0
- obtenirCase(modifierCase(d, j, x, y), x2, y2) =
  - j si x = x2 et y = y2
  - obtenirCase(d, x2, y2) sinon

PARTIE

Sorte :

- Partie

Utilise :

- Joueur
- int
- char * (chaîne de caractère)

Constructeurs

- init : int x Joueur → Partie
- placerPion : Partie x int x int → Partie

Opérateurs :

- aQuiDeJouer : Partie → Joueur
- largeurDamier : Partie → int
- obtenirCase : Partie x int x int → Joueur
- obtenirDamier : Partie → Joueur[]
- obtenirTour : Partie → int
- quiGagne : Partie → Joueur
- sauvegarder : Partie x char * → __
- charger : char * → Partie
- libererMemoire : Partie → __

Invariant :

- aQuiDeJouer(p) != J0

Préconditions :

- placerPion(p, x, y) → 0 ≤ x, y < largeurDamier(p) ^ obtenirCase(p, x, y) = J0
- obtenirCase(p, x, y) → 0 ≤ x, y < largeurDamier(p)

Axiomes :

- aQuiDeJouer(init(l, j)) = j
- aQuiDeJouer(placerPion(p, x, y)) =
    - J1 si aQuiDeJouer(p) = J2,

    - J2 sinon
- largeurDamier(init(l, j)) = l
- largeurDamier(placerPion(p, x, y)) = largeurDamier(p)
- obtenirCase(init(l, j)) = J0
- obtenirCase(placerPion(p, x, y), x2, y2) =
    - aQuiDeJouer(p) si x = x2 et y = y2
    - obtenirCase(p, x, y) sinon
- obtenirTour(init(p)) = 0
- obtenirTour(placerPion(p, x, y)) = obtenirTour(p) + 1
- quiGagne(init(l, j)) = J0
- quiGagne(placerPion(p, x, y)) =
    - aQuiDeJouer(p) si le coup en (x,y) est gagnant
    - J0 sinon
- sauvegarder(p1, str) ^ p2 = charger(str) → p1 identique à p2

MINIMAX

Sorte :

- MiniMax

Utilise :

- Damier
- int

Constructeurs :

- init : Damier x MiniMax → MiniMax
- ajouterFils : MiniMax x MiniMax x int → MiniMax

Opérateurs :

- pere : MiniMax → MiniMax
- nbFils : MiniMax → int
- fils : MiniMax x int → MiniMax
- enleverFils : MiniMax x int → MiniMax
- damier : Minimax → Damier
- ***valeur : MiniMax → int***
- libererMemoire : MiniMax → __

Préconditions :

- fils(mnx, i) → 0 < i ≤ nbFils(mnx)
- enleverFils(mnx, i) → 0 < i < nbFils(mnx)

Axiomes :

- pere(init(d, papa)) = papa
- pere(ajouterFils(mnx, fils, i)) = pere(mnx)
- nbFils(init(d, papa)) = 0
- nbFils(ajouterFils(mnx, fils, i)) = nbFils(mnx) + 1
- fils(ajouterFils(mnx, fils, i), j) =
  - fils si i = j
  - fils(mnx, j) si i > j
  - fils(mnx, j-1) si i < j

- enleverFils(ajouterFils(mnx, fils, i), j) =
  - mnx si i = j
  - enleverFils(mnx, j) si i > j
  - enleverFils(mnx, j-1) si i < j
- damier(init(d, papa)) = d
- damier(ajouterFils(mnx, fils, i)) = damier(mnx)
- ***valeur(mnx) =***
  - 1 si nbFils(mnx) = 0
  - -1 si il existe i : 0 < i ≤ nbFils(mnx) ^ valeur(fils(mnx, i)) = 1
  - 1 sinon (i.e. pour tout i, valeur(fils(mnx, i)) = -1)

*Note : la fonction **valeur** représente tout l'intérêt de la structure : elle renvoie +1 pour les coups gagnants, -1 pour les autres (i.e. les coups perdants)*

Le TAD de cette structure n'est pas définitif. Cela constitue le travail de la semaine prochaine.