MASTER EN SCIENCES INFORMATIQUES

Conception et implémentation d'un générateur de jeux éducatifs

Jacques, Eric

Award date:
1996

Awarding institution:
Universite de Namur

Année Académique 1995-1996.

"Conception et implémentation d'un générateur de jeux éducatifs"

Eric JACQUES

Promoteur : **Monsieur Claude CHERTON**

Mémoire présenté en vue
de l'obtention du grade de Licencié
et Maîte en informatique

Résumé

Le but de ce mémoire est de présenter l'analyse et l'implémentation d'un générateur de jeux éducatifs. Ce mémoire constitue le prolongement de celui effectué par Yves Lowette et Jean-Christophe Marchal.

J'ai poursuivi trois objectfis partiellement contradictoires pendant toute la durée de ce travail :

- concevoir et terminer l'implémentation du générateur
- concevoir un logiciel aussi simple que possible à utiliser. La simplicité d'utilisation a été recherchée aussi bien dans l'analyse que dans l'implémentation.
    - au niveau de l'analyse, en s'efforçant de simplifier au maximum le formulaire de définition des règles du jeu, en s'efforçant de manipuler des concepts simples pouvant êtres compris par des personnes non familiarisées avec l'informatique
    - au niveau de l'implémentation où un effort particulier a été porté sur l'interface
- offrir de larges possiblités de création de jeux

Abstract

The purpose of this thesis is to present the analysis and the implementation of an educational games generator. This continues the work done by Yves Lowette and Jean-Christophe Marchal.

I had three main objectives :

- To conceive and complete the games generator.
- To create a software as easy to use as possible. The simplicity to use was searched for, as well the analysis and the implementation of the generator.
    - analysis : the form of game rules is very simple and anyone not used to computers can understand what it is about.
    - implementation : The interface part of the implemention is conceived to be very easy to use.
- To offer as many possibilities as possible to games creations.

Remerciements

C'est pour moi un plaisir d'exprimer ici mes sincères remerciements à Monsieur le Professeur Claude CHERTON pour sa disponibilité et surtout pour sa patience.
Je ne pense pas que je serais arrivé au bout de ce mémoire sans son aide précieuse.

1. Introduction

Ce mémoire constitue le prolongement et l'accomplissement du travail présenté par Yves Lowette et Jean-Christophe Marchal et qui s'intitulait "Conception d'un générateur de jeux éducatifs". Il se propose de vous présenter un générateur de jeux éducatifs tant du point de vue de sa conception que de son implémentation. Ce qui n'avait pas été réalisé auparavant.

Objectif

1. Un logiciel simple

Le générateur de jeux éducatifs est destiné à des éducateurs s'occupant de personnes ayant un handicap mental léger. Il leur permettra de créer des jeux adaptés aux personnes dont ils ont la charge. Il y a en effet très peu de jeux destinés à ces utilisateurs et correspondant aux difficultés très variées des enfants handicapés.

Les éducateurs étant non professionnels de l'informatique, le logiciel créé sera aussi simple à utiliser que possible. Cet objectif de simplicité sera poursuivi tout au long de l'analyse et de la conception des interfaces.

2. Le contrôle

Les jeux créés sont destinés à des personnes ayant un handicap mental léger. Il convient que le logiciel comporte un contrôle strict des règles du jeu. Des messages d'explication sont envoyés par l'ordinateur lorsque le joueur ne respecte pas les règles du jeu. D'autre part, un message de féliciation apparaît lorsque l'utilisateur gagne le jeu.

Démarche suivie

J'ai repris dans un premier temps la démarche effectuée par Yves et Jean-Christophe, à savoir : j'ai complété leur liste de jeux par d'autres, notemment par des jeux où les jetons se déplacent à l'intérieur de la grille pour déterminer un nouveau paradigme unificateur des jeux étudiés.

J'ai ensuite tenté de simplifier les règles du jeu proposées par Yves et Jean-Christophe, tout en gardant la même sémantique.

A partir du paradigme unificateur et des règles du jeu, j'ai défini un nouveau formulaire de définition des règles du jeu.

Je me suis ensuite consacré à la partie purement informatique du travail, à savoir : la découpe en objets, les services offerts par ces objets, l'analyse des messages envoyés lors du déplacement d'un jeton et l'étude de l'interface proposée.

2. Présentation et Analyse de nouveaux jeux

Dans le but d'enrichir les concepts et les caractéristiques proposées par Yves Lowette et Jean-Christophe Marchal, j'ai complété la série de jeux étudiés par des jeux suivants :

2.1. Le solitaire

But :

Au début du jeu, un ensemble de jetons sont posés sur la grille. Le but est de retirer un à un les jetons jusqu'à ce qu'il n'en reste plus qu'un. On enlève un jeton en sautant par-dessus à l'aide d' un autre.

Règles de déplacement :

- par saut avec prise de la pièce sautée.

Règles de validation :

- lors de la sélection : la pièce est-elle éligible pour un saut ? si oui, validation de la sélection .Dans la négative, annulation de la sélection.

- après le déplacement : s'agit-il d'un saut ? si oui, la pièce sautée est enlevée et éventuellement mise dans la zone de résultat, si non elle est remise à sa position initiale.

Règles de fin de jeu :

Le jeu est fini lorsqu'il ne reste plus qu'une seule pièce.

2.2. Le jeu de dame chinois

But :

C'est un jeu de stratégie se jouant par au moins deux joueurs. Chaque joueur dispose d'un ensemble de jetons rangés dans un endroit de la grille. Le but du jeu est de déplacer ses jetons le plus rapidement possible à l'endroit opposé de la grille. Chaque joueur joue à tour de rôle, un joueur déplace un jeton lui appartenant en le déplaçant d'une case ou en effectuant un (ou plusieurs) saut. Un seul jeton peut se trouver à un moment donné dans une case. Les jetons ne sont pas enlevés en cas de déplacement par saut.

Règles de déplacement :

- déplacement d'une seule case
- par sauts multiples sans prise

Règles de validation :

- Validation de la sélection :
  - le jeton appartient-il au joueur qui a la main ?
    si oui : validation de la sélection
    si non : invalidation de la sélection
  - si le jeton a déjà été sélectionné et a effectué au moins un saut, le jeton est désélectionné et la main passe au joueur suivant.
- Validation du déplacement
  - en cas de déplacement d'une seule case à destination vide sans qu'aucun saut n'ait été effectué par ce jeton pendant le tour de jeu, le déplacement est validé et la main passe au joueur suivant.
  - si le déplacement effectué est un saut et la case de destination est vide, le déplacement est validé, le joueur garde la main, le jeton reste sélectionné pour effectuer un nouveau saut.

Règles de fin de jeu :

Après chaque déplacement d'une pièce, on contrôle si toutes les pièces du joueur ayant la main sont arrivées dans la zone de résultat, si oui, ce joueur est mis en sommeil.

Le jeu est fini lorsque tous les joueurs sont mis en sommeil.

2.3. Jeu de l'oie

But :

Chaque joueur déplace ses pièces en lançant un dé. Le premier joueur qui place sa pièce dans la zone d'arrivée a gagné.

Nouveaux concepts :

- le dé
- les jetons actions
- nouvelles zones de la grille : zone de début et zone de fin.

Règles de* validation *:

- lors de la sélection d'une pièce : chaque joueur doit jouer à son tour, les pièces doivent donc être sélectionnées à tour de rôle. Si la sélection est acceptée, il y a affichage d'un dé (il faut cliquer sur le dé pour le lancer).

- lors du déplacement : il faut que le nombre de cases avancées corresponde au dé.

- après le déplacement : si la case d'arrivée est une case action, exécution de l'action correspondante :

- lancement du dé.
- recul de x cases.
- renvoi de la pièce vers la zone de début.

Règles de fin de jeu :

Le jeu est fini lorsqu' une des pièces arrive dans la zone de fin.

2.4. Casse-tête

But :

Un ensemble de pièces de formes différentes se trouve dans la grille, il faut déplacer une forme particulière d'un endroit à l'autre de la grille (variante du taquet).

Règles de déplacement :

Déplacement de pièces (composées de plusieurs jetons) d'une seule case. Tous les jetons composant la pièce sont déplacés d'une case.

Règles de fin de jeu :

Le jeu est fini lorsqu'une pièce particulière arrive dans la zone de résultat.

2.5. Jeu de stratégie

But :

Chaque joueur place ses jetons dans une zone de la grille. Lorsque tous les jetons sont placés, chaque joueur déplace à tour de rôle un jeton. Lorsque deux jetons (de joueurs différents) se trouvent sur la même case, il y a combat. Le résultat du combat est la comparaison des points gagnés par la pièce attaquante et les points obtenus par la pièce défenderesse. En cas d'égalité, soit les deux pièces sont enlevées, soit il y a tirage au sort du vainqueur.

Règles de validation :

- lors de la sélection : chaque joueur joue une pièce à tour de rôle .Le programme vérifie que la pièce jouée corresponde au joueur qui a la main.

- lors du déplacement : vérification et controle du déplacement. Chaque pièce a ses propres règles de déplacement.

- après le déplacement :

  - si la case d'arrivée est occupée par une pièce du même joueur, le déplacement est refusé

  - si la case est vide, le déplacement est accepté:

  - si la case est occupée par une pièce d'un autre joueur, il y a combat (contrôle entre les points d'attaque de la pièce qui se déplace et les points de défense de la pièce immobile).

Règles de fin de jeu :

Le jeu est fini lorsqu'il n'y a plus qu'un seul joueur en jeu.

3. Modifications apportées aux concepts

3.1. Objectifs

Avant de parler des concepts manipulés, il est important de déterminer quelles sont les caractéristiques que doit posséder le logiciel.

Il faut d'une part que le logiciel soit simple d'utilisation. Le travail effectué par Yves et Jean-Christophe a été de simplifier la définition des règles du jeu (partie la plus complexe lors de la création d'un jeu). Ils proposent comme outil de définition des règles du jeu, l'utilisation d'un formulaire.

Le formulaire est composé d'une série de questions auxquelles il faut répondre par oui,par non ou par une série de valeurs. Lorsque le formulaire est rempli, les règles du jeu sont définies.

Il faut que le logiciel soit modulable. Une modification apportée à un concept ne doit pas affecter l'ensemble du logiciel. Si on construit un logiciel modulable, on peut se permettre dans un premier temps de laisser certains concepts de côté, en sachant qu'il ne sera pas trop compliqué de les intégrer par la suite au logiciel.

Un objectif non négligeable du logiciel est sa capacité à générer un vaste éventail de jeux. Cela n'a en effet pas de sens de créer un générateur de jeux qui ne peut que générer des variantes du jeu de mémoire.

Enfin, un objectif est lié au public visé par le logiciel. Il s'agit de personnes ayant un handicap mental léger. Il est important que le logiciel contrôle le respect des règles du jeu. Il faut que éventuellement le logiciel affiche un message d'explication en cas d'erreur, un message d'encouragement en cas de réussite ...

Un moyen utilisé pour rencontrer l'objectif de simplicité d'utilisation est l'emploi généralisé de valeurs par défaut pour toutes les caractéristiques des concepts utilisés. Le concepteur peut par exemple créer les jetons d'un jeu, la grille, placer les jetons dans la grille et tester le jeu sans même définir les règles du jeu. Il pourra à tout moment, sans quitter le jeu, modifier la règle du jeu, changer les caractéristiques des jetons, et continuer à jouer pour voir les effets de ces modifications sur le jeu en cours.

3.2. Nouveaux concepts

Malgré l'objectif de simplicité d'utilisation, qui sous entend, utiliser un minimum de concepts et un minimum de caractéristiques pour ces concepts, j'ai décidé de rajouter une série de concepts à la liste des concepts utilisés.

Notion de joueur et de tour de jeu

Il ne faut pas oublier que les jeux sont destinés à être joués par des communautés de personnes. L'intérêt de jouer à plusieurs personnes sur le même jeu est un aspect qu'il ne faut pas négliger.

Cette notion entraîne une complexification de la génération du jeu : concept de joueur et de tour de jeu. Ces concepts sont tellement utilisés dans le monde du jeu que leur signification est intuitive à tout concepteur.

La plupart des jeux présentés peuvent se jouer à plusieurs joueurs. Le programme indique dans la zone de message le nom du joueur à qui c'est le tour de jeu (le joueur qui « a la main »). Le concepteur doit définir pour chacun des déplacements suivants si la main passe au joueur suivant : un déplacement valide, un déplacement non valide. La main peut changer également à la demande du joueur.

Le concepteur d'un jeu détermine le nombre de joueurs pouvant jouer à ce jeu. Si le concepteur ne crée qu'un seul joueur, ce jeu pourra être joué par un nombre quelconque de joueurs. Par contre, si le concepteur crée deux ou plusieurs joueurs, le jeu devra être joué par le nombre exact de joueurs prévus.

Outre la possibilité de pouvoir jouer à plusieurs joueurs sur le même jeu, la notion de tour de jeu élargit l'éventail des jeux pouvant être générés. Prenons pour exemple le jeu de mémoire :

⇒ si la main change uniquement lors d'un déplacement valide, le joueur gardera la main tant que les sélections qu'il effectue ne sont pas validées. Le but ici n'est pas de prendre le plus de jetons mais bien de faire le moins d'erreurs possibles.

⇒ si la main change uniquement lors d'un déplacement non valide, le but du jeu est de collecter le plus grand nombre de jetons. Si un joueur effectue une association valide il garde la main. Le but ici est de prendre le plus de jetons. Dans cette variante, lorsqu'un joueur effectue sa dernière sélection, il a intérêt à choisir un jeton qu'il connaît déjà de manière à ne pas donner de trop grandes chances aux adversaires (le joueur qui a la main la garde tant qu'il effectue des associations valides).

⇒ si la main change lors d'un déplacement valide et lors d'un déplacement non valide, chaque joueur joue à son tour.

⇒ si la main change à la demande des joueurs, le jeu devient un jeu de stratégie : commettre le moins d'erreurs possibles tout en collectant le plus de jetons

Lorsqu'un joueur a effectué un déplacement, il y a vérification des conditions de fin de jeu. Ces conditions sont doubles : conditions entraînant le gain de la partie ou conditions entraînant la perte du jeu. Lorsqu'un joueur a gagné ou perdu il est mis en sommeil, les autres joueurs peuvent continuer à jouer si le concepteur a fait ce choix dans le formulaire de règles du jeu.

Dans certains cas cependant, tous les joueurs sont mis en sommeil. Ces conditions sont : la grille vide ou la grille remplie. Lorsque le concepteur indique la fin du jeu , tous les joueurs sont mis en sommeil dès que la grille est vide.

Le dé

Le dé est un objet comportant un certain nombre de faces. A chacune des faces du dé est liée une action. Lorsqu'un dé est lancé, il y a tirage au sort d'une face du dé et exécution de l'action liée à la face.

C'est un concept qui me tient à coeur. Il permet non seulement de déplacer les jetons dans la grille mais peut être utilisé comme un outil de définition des règles du jeu, comme nous le verrons par la suite.

Lors de la conception d'un dé, le concepteur définit :

- le nombre de faces du dé, l'action liée à chacune des faces

- le nombre de faces à afficher et la durée d'affichage d'une face lors du lancement du dé. Si le nombre de faces à afficher est 0, le dé n'est pas visible et l'action est exécutée immédiatement. Si ce nombre est x, il y a x tirages au sort d'une face, chaque face est affichée pendant un certain temps. L'action liée à la dernière face affichée est exécutée.

Il existe deux types de dés : le dé lié à un jeton et le dé lié à une case.

- Dé lié à un jeton: Prenons le cas du jeu de l'oie, les jetons sont déplacés par lancement d'un dé à 6 faces. L'action liée à la face 1 est un déplacement de +1 case, l'action liée à la face 2 est un déplacement de +2 cases, etc. ... Lorsqu'un joueur sélectionne un jeton pour lequel on a défini un dé, celui-ci apparaît à l'écran. S'il désélectionne le jeton, le dé disparaît. Si il clique sur le dé, le dé est lancé et l'action liée au dé est exécutée.

- Dé lié à une case: C'est un dé pour modéliser une action particulière à effectuer lorsqu'un jeton se pose dans une case de la grille. Si un jeton se pose dans une case dans laquelle il y a un dé, ce dé est lancé automatiquement. Exemple d'un dé lié à une case tiré du jeu de l'oie : dé utilisé pour effectuer l'action « recule de 3 cases » lorsqu'un jeton se pose dans une case, le concepteur place dans cette case un dé à une face. L'action liée à la face du dé est déplacement de -3 cases.

Les actions pouvant êtres liées à une face de dé sont :

- Déplacement « automatique » du jeton de x cases. Si x est positif, le jeton avance de x cases, si x est négatif le jeton recule de x cases. Un déplacement automatique est un déplacement effectué par le programme. Il est à noter que si le concepteur utilise un dé de type déplacement il doit impérativement numéroter les cases de la grille. Pour simplifier la

tâche du concepteur, lorsque le concepteur définit un dé de type déplacement, il y a numérotage par défaut des cases de la grille.

- Déplacement « manuel » du jeton de x cases. Le joueur doit déplacer son jeton de x cases, tout autre action du joueur sera refusée. Après y tentatives, le jeton est déplacé automatiquement.
- Renvoi du jeton vers le sabot.
- Renvoi du jeton vers la zone de résultat.
- x lancement(s) d'un autre dé.
- Elimination du jeton.
- La main passe au joueur suivant.
- Mise en sommeil du joueur propriétaire du dé.

3.3. Modifications apportées aux concepts

Le jeton

Yves Lowette et Jean-Christophe Marchal proposaient une série de caractéristiques des jetons. Pour chacune de ces caractéristiques, je me suis posé la question suivante : cette caractéristique est-elle vraiment indispensable pour créer les jeux étudiés ? Pour optimiser l'objectif de simplicité, il faut veiller à utiliser le moins de concepts possibles pour la création des jeux. Il faut tenter de retirer des jeux étudiés, les caractéristiques minimums, indispensables pour créer la majorité des jeux étudiés.

Les caractéristiques suivantes étaient définies par Yves Lowette et Jean-Christophe Marchal :

Identifiant d'un type de jeton. Il s'agit d'aider le concepteur dans sa tâche de création des jetons. Lorsque l'on analyse les différents jeux, on se rend compte que la plupart des jetons créés utilisent des caractéristiques semblables. L'idée proposée est de sauver ces caractéristiques communes à plusieurs jetons dans un type de jeton. Le concepteur désirant créer un jeton pourra s'il le désire indiquer que ce jeton hérite des caractéristiques stockées dans un type de jeton.

Cette caractéristique est-elle intuitive à un éducateur ? Je ne pense pas que l'idée de créer un type de jeton abstrait gardant trace de caractéristiques couramment utilisées soit vraiment intuitive pour une personne non familiarisée avec l'informatique. Cette idée favorise-t-elle la création d'un plus large éventail de jeux ? Non, elle vise à simplifier la tâche du concepteur. Cette simplification est faite au détriment de la simplicité d'utilisation du logiciel.

Il faut en effet, créer une librairie des types de jetons, offrir des outils permettant de créer, modifier et hériter les caractéristiques des jetons, gérer les conflits entre les caractéristiques des types de jetons et les caractéristiques définies dans les jetons, ..

Pour toutes ces raisons, je n'ai pas gardé la caractéristique identifiant d'un type de jeton. Cette idée me semblait intéressante à creuser. Lorsque l'on y réfléchit, il s'agit d'une aide à offrir au niveau de l'interface du programme et non une caractéristique d'un jeton. Mon idée est proche de celle du type de jeton tout en évitant de créer un niveau d'abstraction supplémentaire. Puisque la notion de type de jeton est basée sur le fait que l'on désire créer un grand nombre de jetons avec des caractéristiques fort similaires, on peut imaginer que de créer le premier jeton avec ces caractéristiques n'implique pas plus de travail au concepteur d'un jeu. Il suffit d'offrir au concepteur la possibilité de copier les caractéristiques du jeton actuellement en cours d'édition dans le nouveau jeton qu'il désire créer, pour que le jeton en cours d'édition serve de type de jeton.

Les attributs position de départ et position d'arrivée peuvent être éliminés sans nuire aux possibilités de créations des jeux. Pour plus d'explication à ce sujet, lire la partie règles d'associations du chapitre règles du jeu. La diminution du nombre de concept utilisés sans nuire aux possibilités de création de jeux et à la facilité d'utilisation du logiciel est favorable à la fois aux objectifs concepteur et joueur.

Le concept de pièce a également été supprimé. Je suis convaincu que l'implémentation de ce concept apporterait un plus à un générateur de jeu. Il y a une foules de jeux utilisant le concept de pièce (parmi les jeux étudiés : les dominos ou le casse-tête). Bien que cette notion soit intuitive pour le concepteur elle semble également fort difficile à implémenter. De manière à ne pas mettre l'entièreté du travail en péril (les délais pour remettre le travail ne sont pas extensibles), j' ai décidé de ne pas considérer la notion de pièce et de laissé de côté les caractéristiques liées a ce concept , à savoir : pivotable et réalisable par le joueur.

La caractéristique mobile dans la grille (objet pouvant se déplacer dans la grille selon un parcours fixé par le concepteur), me semblait hors de propos, je l'ai également laissé tombée.

Il ne reste donc des caractéristiques proposées que :

- superposables: lorsqu'un jeton est superposable, un ou des autres jetons peuvent êtres posés sur lui dans la même case.

- listes de caractéristiques du jeton. A chaque jeton peut être associé une liste de caractéristiques le décrivant. Cette liste constituée par le concepteur est utilisée par le générateur pour le contrôle des règles du jeu. Par exemple, pour le jeux des familles, le concepteur peut concevoir une famille de jetons représentants des animaux volants. Il donne à chacun de ces jetons la caractéristique « vol ». Il peut également concevoir une série de jetons représentant des animaux non volants à qui il donne la caractéristique « non volant ». Les caractéristiques des jetons sont utilisées pour vérifier si le joueur a appareillé correctement les différents jetons.

Nouvelles caractéristiques ajoutées au jeton :

- Ajout dans la liste des caractéristiques d'un jeton spécial identifiant le (les) propriétaire(s) du jeton. Comme je l'ai déjà mentionné, il est important de créer des jeux pouvant êtres joués à plusieurs vu le public ciblé. Une caractéristique importante des jetons dans les jeux multi-joueur est le propriétaire du jeton. Dans certains jeux, celui-ci est la seule personne à pouvoir le déplacer. Cette caractéristique correspond à une notion intuitive du concepteur et vise à renforcer l'objectif du contrôle du respect des règles du jeu par le joueur. Le fait de déterminer si un jeton appartient à un joueur se fait de deux manières :

  ⇒ automatique : lorsque le concepteur demande à un sabot d'un joueur de « marquer le propriétaire du jeton »,le sabot ajoute automatiquement à la liste des caractéristiques du jeton, l'identifiant du joueur propriétaire du sabot. Il faut donc faire attention dans ce mode, si le concepteur autorise le joueur à déplacer le jeton de la grille vers le sabot. Lorsque le joueur ayant la main déplace le jeton vers le sabot d'un autre joueur, plusieurs joueurs peuvent devenir propriétaires du jeton.

  ⇒ manuelle, c'est le concepteur qui ajoute l'identifiant du joueur comme caractéristique du jeton. Un jeton peut avoir plusieurs caractéristiques « identifiant du propriétaire du jeton ».

- Combat. Les jetons de type combat ont deux valeurs associées qui sont les points d'attaque et de défense. Lorsque le jeton d'un joueur se pose dans une case occupée par un jeton du joueur adverse il y a combat, c'est à dire comparaison entre les points d'attaque de la pièce se déplaçant et les points de défense du jeton de la case de destination. Les jeux utilisant cette caractéristique sont les jeux du type « stratego ». Etant donné la faible proportion de jeux utilisant cette caractéristique et le bouleversement que la prise de cette caractéristique entraînerait, j'ai abandonné l'idée de l'incorporer dans le générateur de jeux.

- Déplacement. Ajout d'une caractéristique déplacement au jeton décrivant quels sont les déplacements autorisés pour un jeton. Cette caractéristique me semble importante, plutôt que de définir les déplacements autorisés dans les règles du jeu, il me semble plus intéressant de l'introduire comme caractéristique du joueur. Si on définit les déplacements comme caractéristique des règles du jeu, tous les jetons doivent se déplacer de la même manière, ce qui n'est pas forcément le cas. Prenons l'exemple de l'arithmétique, le concepteur place des jetons dans la grille. Ces jetons sont immobiles et ne pourront pas êtres manipulés par le joueur. D'autres jetons sont placés dans le sabot, ceux-ci pourront être déplacés du sabot vers la grille et de la grille vers le sabot.

- Reprenons la liste des déplacements possibles à l'intérieur de la grille des jeux étudiés : déplacement par lancement d'un dé, par un saut sans prise de jeton, par des sauts multiples sans prise de jeton, par un saut avec prise de jeton, par des sauts multiples avec prise de jeton, par un déplacement de maximum x cases, par une combinaison de ces déplacements. Exemple de combinaison de déplacements : le jeu de dame chinois autorise un déplacement d'une case ou par sauts multiples sans prise.

La grille

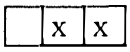
La grille représente l'espace de jeu, elle est obligatoire. Chaque case peut recevoir un ou plusieurs jetons. Les cases sont toutes de formes identiques (carrées ou rectangulaires). Un espace fixe peut être laissé entre les lignes et/ou les colonnes.

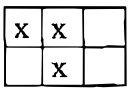
La grille est placée sur un dessin de fond, elle est visible ou non. Il est possible de placer des jetons de type dessin dans la grille. Un jeton de type dessin est un jeton placé dans une, plusieurs ou toutes les cases de la grille pour y placer un dessin.

La grille peut contenir un dessin, le dessin contenu dans la grille a toujours la même taille que la grille.

Nouvelles caractéristiques de la grille

- espace entre deux colonnes
- espace entre deux lignes
- hauteur des cases
- largeur des cases
- épaisseur du trait
- tableau des déplacements possibles (voir cases de la grille)
- Zones. La grille est divisée en zones. Il n'y a plus de distinction entre zones libres ou fixées. Une zone est un ensemble de cases de la grille. Cet ensemble de cases possède une ou plusieurs caractéristiques. Ces caractéristiques sont soit « caractéristiques concepteur » c'est-à-dire des caractéristiques fixées par le concepteur lors de la création de la zone ou « caractéristiques joueur », c'est-à-dire des caractéristiques fixées par le joueur. Un joueur donne des caractéristiques à une zone en y plaçant un ou plusieurs jetons.
- Cases de la grille.
    - Dé posé dans la case (cfr. supra).
    - Numéro de la case. La numérotation des cases est utilisée pour les jeux se jouant avec un dé de type déplacement.
    - tableau des déplacements possibles (dans une matrice 3 sur 3). Exemple :





les x représentent les mouvements possibles à partir de la case courante. Le x du centre indique que la case est éligible pour un déplacement. Cette notion de déplacement possible à partir d'une case est considérée comme une notion intuitive pour les concepteurs de jeux. Pour ne pas obliger le concepteur à définir les déplacements pour chaque case de la grille, un déplacement par défaut peut être défini au niveau de la grille. Si aucune table de déplacement n'a été définie pour une case, c'est le tableau de déplacement de la grille qui est utilisé lors du contrôle des jetons se déplaçant d'une seule case. Si un mouvement positionne le jeton hors de la grille, il y a roulement de sorte que le jeton se positionne de l'autre côté de la grille.

Les attributs position de la grille, espace entre deux lignes, entre deux colonnes, hauteur des cases, largeur des cases, sont liés à l'interface. Ces attributs sont cachés du concepteur. Ils sont générés par l'interface. Lorsque le concepteur déplace la souris, augmente la taille des cases, ... l'interface stocke les valeurs actuelles des attributs sans que le concepteur soit conscient de leurs existences. Leur présence vise à aider le concepteur à placer de manière adéquate la grille sur l'image de fond.

Le sabot

Un sabot est un endroit où l'on stocke les jetons qui vont êtres présentés au joueur. Dans certains jeux, l'entièreté du sabot est affichée à l'écran, dans d'autres non. Pour pouvoir définir l'ensemble des jetons à afficher dans le sabot, il faut un endroit pour stocker les jetons en attente, c'est ce que nous allons appeler la file d'attente du sabot.

Si le concepteur du jeu crée un seul joueur et un seul sabot, ce sabot sera utilisé pour tous les joueurs, si le concepteur définit plusieurs joueurs, c'est le concepteur qui décide si un sabot est utilisé par un ou par plusieurs joueurs. De nouveau, cette notion est cachée au sein de l'interface.

Dans les jeux multi-joueurs, le concepteur peut choisir d'utiliser un seul sabot pour tous les joueurs, ou au contraire créer plusieurs joueurs et donner un sabot à chaque joueur.

Caractéristiques du sabot : Ces caractéristiques correspondent à la définition intuitive que le concepteur se fait d'un sabot.

- propriétaire du sabot
- option de remplissages : à la demande de l'utilisateur, dès qu'une case se libère, lorsque le sabot est vide
- nombre de jetons se trouvant dans la file d'attente

- ordre de remplissage du sabot, faut-il garder l'ordre des jetons se trouvant dans la file d'attente ou faut-il remplir le sabot de manière aléatoire avec des jetons provenant de la file d'attente ?

La zone de résultat

Il y a une ou plusieurs zones de résultat (une pour tous les joueurs ou une par joueur). Le concepteur peut choisir l'ordre de remplissage de la zone de résultat : la zone est remplie ligne par ligne ou colonne par colonne.

Le concepteur doit déterminer la manière de remplir la zone de résultat : remplissage par ligne ou par colonne. Le concepteur doit également indiquer les actions à effectuer lorsque la zone est remplie :

⇒ décaler la zone d'une ligne

⇒ décaler la zone d'une colonne

⇒ vider la zone de résultat

⇒ arrêter de mettre les jetons enlevés du jeu dans la zone de résultat

4. Les règles du jeu proposées

C'est la partie la plus importante et la plus complexe du travail de conception. Les règles du jeu sont constituées des règles de validation des déplacements ainsi que des règles de détection de la fin du jeu.

4.1. Rappel des règles proposées

Les règles du jeu sont composées d'un formulaire dans lequel il suffit de cocher des choix. Les règles du jeu sont composées des règles d'association, des règles de validation, des règles de déplacement et des règles de fin de jeu.

Les règles d'association

Les jeux étudiés peuvent êtres rangés en trois catégories : association des jetons par caractéristique, par côtés ou par position.

Association par côtés

Il est possible de définir le(s) côté(s) d'un objet qui peut être accolé à un ou plusieurs autres. C'est le cas du jeu de dominos.

Association par caractéristiques

A chaque jeton est associé une liste de caractéristiques le décrivant. C'est le cas du jeu de mémoire.

Association par position

Un jeton peut être associé à au moins une position sur la grille. Il ne pourra occuper qu'une seule des places qui lui est assignée. Tout autre place de la grille sera non valide. C'est le cas du puzzle.

Les règles de validation

Elle permet au concepteur de déterminer les moments où des vérifications de la correction des associations des objets par le joueur seront réalisées, et de définir les actions qui accompagneront cette vérification.

- **moment** : après x coups, x secondes, x minutes, à la demande, sabot vide, ...

- **actions liées** : message d'encouragement, retrait des pièces correctement jouées avec message, retour des pièces incorrectement jouées vers le sabot avec message.

- **taux d'erreur** : lors de chaque validation, on peut modifier la valeur d'un compteur du nombre d'erreurs commises. Au-delà d'un taux fixé, le jeu peut se terminer avec un message.

Manipulation

Détermine les modes de manipulation autorisés sur les jetons. Il existe trois modes de manipulation : soit les sélections seules sont autorisées, soit seul les déplacements sont autorisés, soit les déplacements et les sélections sont autorisés.

Les règles de déplacement

Les règles de déplacement déterminent les mouvements autorisés entre les zones de l'écran. Il s'agit de déterminer si les déplacements suivants sont autorisés :

- de la grille vers la grille, vers le sabot, ou vers la zone d'attente (automatique pour la zone de résultat).

- du sabot vers le sabot, vers la grille ou vers la zone d'attente.

- de la zone d'attente vers la zone d'attente, vers la grille ou vers le sabot.

Les règles de fin de jeu

Pour la grille, le sabot et la zone d'attente, le concepteur doit déterminer si elles doivent être vides ou remplies à la fin du jeu (indépendemment les unes des autres).

Le jeu se termine lorsque, soit la grille est remplie ou vide, soit la grille et le sabot sont vides, le temps imparti est écoulé, le nombre maximum d'erreur est atteint.

5. Modifications apportées aux règles du jeu

Par l'étude des nouveaux jeux, j'ai tenté d'élargir le spectre de jeux pouvant être générés à une catégorie importante des jeux de plateaux : les jeux se jouant uniquement sur une grille et dans lesquels les jetons se déplacent à l'intérieur de la grille. Ces jeux sont caractérisés par des modes de déplacement variés (parfois complexe) des jetons : déplacement par dé, déplacement libre, déplacement d'une case, de x fois une case, de x cases, de x fois x cases, déplacement par saut, par sauts multiples, par saut avec prise, par sauts multiples avec prise, ou une combinaison de ces modes de déplacement (par exemple le jeu de dame chinois).

Il faut cependant éviter que la conception des jeux ne devienne une tache complexe tout en essayant d'élargir les possibilités de création des jeux. Il n'est donc pas question de réaliser un générateur de jeux de plateau dont l'utilisation serait à ce point complexe qu'elle demanderait une longue période de formation.

La définition des règles du jeu est la partie la plus abstraite dans la conception d'un jeu et est également la partie la plus complexe. Je me suis demandé comment simplifier le formulaire de définition des règles du jeu.

Cette simplification a été opérée à deux niveaux :

- au niveau des règles du jeu, j'ai simplifié les règles du jeu proposées en diminuant le nombre de caractéristiques (dans la mesure du possible sans perte de possibilités de création de jeux).

- en déplaçant une partie importante des concepts du formulaire de définition des règles du jeu vers l'interface. La gestion par l'interface de certains concepts permet de cacher au concepteur certaines caractéristiques (par exemple coordonnée de la grille) ou permet d'en supprimer d'autres (par exemple identifiant).

Les règles d'association

Je vais présenter chacune des règles proposées et présenter les modifications apportées à ces règles :

Association par côtés

Etant donné les simplifications apportées : pas de concept de pièce, pas de caractéristique « retournable » et l'association par côté perd une grande partie de son intérêt. Je n'ai donc pas repris cette association.

5. Modifications apportées aux règles du jeu

Par l'étude des nouveaux jeux, j'ai tenté d'élargir le spectre de jeux pouvant être générés à une catégorie importante des jeux de plateaux : les jeux se jouant uniquement sur une grille et dans lesquels les jetons se déplacent à l'intérieur de la grille. Ces jeux sont caractérisés par des modes de déplacement variés (parfois complexe) des jetons : déplacement par dé, déplacement libre, déplacement d'une case, de x fois une case, de x cases, de x fois x cases, déplacement par saut, par sauts multiples, par saut avec prise, par sauts multiples avec prise, ou une combinaison de ces modes de déplacement (par exemple le jeu de dame chinois).

Il faut cependant éviter que la conception des jeux ne devienne une tache complexe tout en essayant d'élargir les possibilités de création des jeux. Il n'est donc pas question de réaliser un générateur de jeux de plateau dont l'utilisation serait à ce point complexe qu'elle demanderait une longue période de formation.

La définition des règles du jeu est la partie la plus abstraite dans la conception d'un jeu et est également la partie la plus complexe. Je me suis demandé comment simplifier le formulaire de définition des règles du jeu.

Cette simplification a été opérée à deux niveaux :

- au niveau des règles du jeu, j'ai simplifié les règles du jeu proposées en diminuant le nombre de caractéristiques (dans la mesure du possible sans perte de possibilités de création de jeux).

- en déplaçant une partie importante des concepts du formulaire de définition des règles du jeu vers l'interface. La gestion par l'interface de certains concepts permet de cacher au concepteur certaines caractéristiques (par exemple coordonnée de la grille) ou permet d'en supprimer d'autres (par exemple identifiant).

Les règles d'association

Je vais présenter chacune des règles proposées et présenter les modifications apportées à ces règles :

Association par côtés

Etant donné les simplifications apportées : pas de concept de pièce, pas de caractéristique « retournable » et l'association par côté perd une grande partie de son intérêt. Je n'ai donc pas repris cette association.

Lorsqu'aucun jeton n'a de caractéristique, il n'y a pas de contrôle des règles d'association. Ceci est fait de manière implicite sans devoir obliger le concepteur à devoir se préoccuper des règles d'association.

Les règles de validation

La simplification des règles d'association entraîne automatiquement une simplification des règles de validation. Rappelons que les règles de validation proposaient une validation « retardée » de la validation des mouvements effectués avec une remise à la position initiale des jetons non validés. La validation était effectuée après x déplacements, x secondes ou à la demande du joueur.

Désormais, lorsqu'un jeton se pose dans une zone, cette zone acquiert les caractéristiques du jeton. La validation du déplacement doit se faire avant de poser le jeton dans la zone.

Nouvelles règles de validation

Les règles de validation ne sont désormais plus pertinentes. Cette simplification est-elle compatible avec les objectifs que nous nous sommes fixés ?

L'objectif concepteur est respecté car il y a simplification des concepts utilisés.

Cette simplification ne nuit pas à l'éventail de jeux pouvant êtres générés par le logiciel. De plus comme je l'ai déjà signalé, vu le public ciblé, une validation immédiate des déplacements avec éventuellement un message d'explication est préférable.

Modifications des manipulations

Yves et Jean-Christophe proposaient un mode de sélection "global" des jetons. Le mode de sélection était déterminé au niveau des règles du jeu pour tous les jetons. Le mode de manipulation d'un jeton pouvait soit être déplacement (entre les échiquiers), sélection ou sélection et déplacement.

Première modification : le mode de manipulation n'est plus déterminé globalement dans les règles du jeu, mais individuellement pour chaque jeton lors de la définition du jeton. Ceci offre deux avantages : cela permet de simplifier le formulaire de définition des règles du jeu, sans nuire aux possibilités du logiciel et cela permet d'autoriser la sélection d'un jeton et le déplacement d'un autre.

Lors de l'analyse des jeux étudiés, aucun jeton ne pouvait être à la fois ou sélectionné ou déplacé. Ceci est du à la différence existant entre les jetons dont le mode de manipulation est sélection ou déplacement.

Un jeton dont le mode de manipulation est "sélection" se retourne lorsqu'il est sélectionné. Si un déplacement était autorisé avec ce jeton, il faudrait lors de chaque sélection demander s'il faut retourner le jeton ou le déplacer.

Lorsqu'un jeton dont le mode de manipulation est "déplacement" devient sélectionné va attendre que le joueur choisisse la case de destination du mouvement.

Pour ces raisons le mode de manipulation d'un jeton est soit : "sélection", soit "déplacement", soit "immobile". Un jeton dont le mode de manipulation est immobile ne peut pas être sélectionné par le joueur, il peut être utilisé par le concepteur pour placer un dessin dans une case ou pour expliquer le jeu.

Règles de fin de jeu

Rappel

Le jeu se termine lorsque :

- la grille est vide, pleine, non pertinent (OK)

- le sabot est vide, plein, non pertinent. Le jeu se termine lorsque le sabot est plein ???

- la zone d'attente est vide, pleine, non pertinente. La zone d'attente est considérée comme une extension du sabot. Si le concepteur désire que le jeu se termine lorsque le sabot est vide, cela signifie lorsque le sabot et la zone d'attente sont vides.

- le temps limite est atteint. Je n'ai pas gardé cette caractéristique qui ne m'a pas semblé essentielle dans aucun des jeux étudiés.

- le nombre limite de coups ou de déplacements est atteint (OK).

Modifications des règles de fin de jeu

Autres événements entraînant la fin du jeu :

- Lorsqu'il y a validation « concepteur » des zones de la grille. C'est-à-dire lorsqu' il y a concordance entre les caractéristiques des jetons se trouvant dans les zones et les caractéristiques définies par le concepteur pour la zone. Il faut rappeler que lors d'un déplacement, le concepteur peut autoriser le joueur à déplacer un jeton même s'il n'y a pas validation des caractéristiques, c'est le cas pour le jeu du taquet.

- Lorsqu'il y a validation « concepteur » et le sabot est vide. Ce critère est utilisé pour le jeu du puzzle.

- Lorsqu'il y a validation « concepteur » et la grille est remplie. Ce critère est utilisé pour le jeu d'arithmétique, de français, de prendre 1 élément parmi x, de prendre n éléments parmi x.

Il peut paraître contradictoire avec les objectifs de proposer tant de critères de fin de jeu. Il est à noter cependant que les règles de fin de jeu sont particulières. Elles ne sont pas destinées à élargir l'éventail des jeux pouvant être générés mais bien de féliciter le joueur lorsqu'il est parvenu à réaliser l'objectif qui lui était demandé. C'est également important de contrôler de manière stricte la concordance des déplacements avec les règles du jeu, il faut également contrôler de manière stricte les conditions de fin de jeu. Cela peut être vexant pour un joueur handicapé de réaliser l'objectif demandé sans avoir une petite récompense : un message de félicitations de la part de l'ordinateur C'est pourquoi j'ai tenu à être complet dans les règles de fin de jeu, ainsi que de séparer les messages de fin de jeu pour cause d'échec (nombre maximum d'erreurs ou de déplacements atteints), des messages de fin de jeu pour cause de réussite.

6. Conception du formulaire

6.1. Le paradigme

L'analyse nous a permis de dégager de l'ensemble des jeux étudiés un ensemble de concepts et de caractéristiques liées à ces concepts.

6.2. Les concepts utilisés

J'ai cherché tout au long de l'analyse à déterminer les caractéristiques d'un programme simple à utiliser permettant de créer un ensemble de jeux éducatifs. Je vais résumer tous ces concepts et caractéristiques dans un tableau. Ce tableau va servir à la conception du formulaire des règles du jeu.

Le tableau de la page suivante présente de manière synthétique tous les concepts et les caractéristiques associés à ces concepts. La première colonne reprend les intitulés des concepts, la seconde reprend les caractéristiques liées à ces concepts et la troisième colonne présente le type des valeurs possibles et la valeur par défaut.

Concepts	Caractéristiques	Valeurs
Jeton	- caractéristiques	- liste de caractéristiques, par défaut aucune caractéristique.
	- superposable	- oui ou non, par défaut non.
	- mode de manipulation	- immobile, sélection déplacement, par défaut déplacement.
	- type de déplacement (si le mode = déplacement)	
	- de la grille vers le sabot	- oui ou non, par défaut oui.
	- de la grille vers z attente	- oui ou non, par défaut oui.
	- du sabot vers z attente	- oui ou non, par défaut oui.
	- de la z attente vers sabot	- oui ou non, par défaut oui.
	- de la z attente vers grille	- oui ou non, par défaut oui.
	- dans la grille	- immobile, tout permis, une case, x cases, par défaut tout permis.
Règles d'association	- validation des caractéristiques	- validation minimum ou maximum, par défaut maximum.
	- action à effectuer lorsque les caractéristiques sont validées	- accepter le déplacement ou enlever le(s) jeton(s), par défaut accepter le déplacement (sélection).
	- action à effectuer lorsque les caractéristiques ne sont pas validées	- accepter ou refuser le déplacement, par défaut refuser le déplacement (sélection).
Jeu Gagné lorsque	- la grille est vide	- oui ou non, par défaut non.
	- la grille est remplie	- oui ou non, par défaut non.
	- le sabot est vide	- oui ou non, par défaut non.
	- toutes les zones sont valides	- oui ou non, par défaut non.
	- zones valides et sabot vide	- oui ou non, par défaut oui.
	- zones valides et grille remplie	- oui ou non, par défaut oui.
	- afficher boîte de dialogue gagné	- oui ou non, par défaut oui.
	- message gagné	- message à afficher lorsque la partie est gagnée, par défaut "Bravo ! vous avez gagné".
Jeu Perdu lorsque	- nombre max. de coups atteints	- nombre, par défaut 0 (= non).
	- nombre max. erreur atteint	- nombre, par défaut 0 (= non).
	- afficher boîte de dialogue perdu	- oui ou non, par défaut oui.
	- message perdu	- message à afficher lorsque la partie est perdue, par défaut "Vous avez perdu".
Grille	- coordonnées du coin supérieur gauche de la grille	deux nombres, par défaut 0,0.
	- nombre de lignes	- nombre, par défaut 0.
	- nombre de colonnes	- nombre, par défaut 0.
	- espace entre lignes	- nombre, par défaut 0.

	- espace entre colonnes - hauteur des cases - largeur des cases - épaisseur du trait - zones - ordre jetons de la grille - table de déplacements - nom du fichier contenant le dessin de la grille	- nombre, par défaut 0. - nombre, par défaut 50. - nombre, par défaut 50. - nombre, par défaut 1. - liste de zones, par défaut une zone comprenant toutes les cases de la grille. - ordre aléatoire, garder ordre, par défaut ordre aléatoire. - indiquer, pour chaque direction, si ce déplacement est autorisé ou non. - chaîne de caractère, par défaut vide.
Case de grille	- table de déplacements - numéro	- table de déplacement particulière liée à la case. - nombre, par défaut 0.
Zones	- cases appartenant à la zone - caractéristiques « concepteur »	- liste de cases, par défaut aucune. - liste de caractéristiques, par défaut aucune.
Sabot	- coordonnées du coin supérieur gauche du sabot - présent - nombre de lignes - nombre de colonnes - espace entre lignes - espace entre colonnes - hauteur des cases - largeur des cases - épaisseur du trait - option de remplissage - nom du fichier contenant le dessin du sabot	- deux nombres, défaut 0, 0 - oui ou non, par défaut non. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 50. - nombre, par défaut 50. - nombre, par défaut 1. - un jeton quitte le sabot, sabot vide, à la demande du joueur, défaut lorsqu'un jeton quitte le sabot - chaîne de caractère, par défaut vide.
File d'attente Sabot	- nombre de cases - ordre jetons	- nombre, par défaut 20. - ordre défini, ordre aléatoire, par défaut ordre aléatoire.
Zone de Résultat	- coordonnées du coin supérieur gauche de la zone de résultat - présente - nombre de lignes - nombre de colonnes - espace entre lignes - espace entre colonnes - hauteur des cases - largeur des cases - épaisseur du trait	- deux nombres, par défaut 0, 0 . - oui ou non, par défaut non. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 50. - nombre, par défaut 50. - nombre, par défaut 1.

	- remplissage - zone remplie -nom du fichier contenant le dessin de la zone de résultat	- par ligne ou par colonne, par défaut par ligne. - décaler ligne, décaler colonne, vider zone, arrêter de remplir, par défaut vider zone . - chaîne de caractère, par défaut vide.
Zone d'Attente	- coordonnées du coin supérieur gauche de la zone d'attente - présente - nombre de lignes - nombre de colonnes - espace entre lignes - espace entre colonnes - hauteur des cases - largeur des cases - épaisseur du trait - nom du fichier contenant le dessin de la zone d'attente	- deux nombres, par défaut 0, 0. - oui ou non, par défaut non. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 0. - nombre, par défaut 50. - nombre, par défaut 50. - nombre, par défaut 1. - chaîne de caractère, par défaut vide.
Tour de Jeu	- boîte de dialogue, joueur suivant; présente ? - la main passe quand le déplacement est non validé - la main passe quand le déplacement est validé	- oui ou non, par défaut non. - oui ou non, par défaut oui. - oui ou non, par défaut oui.
Joueur	- nom du joueur - sabot du joueur - zone attente - zone de résultat	- string, par défaut "Joueur". - oui ou non, par défaut non. - oui ou non, par défaut non. - oui ou non, par défaut non.
Dé	- liste de faces - durée d'affichage d'une face - nombre de faces à afficher	- défaut aucune face. - nombre, défaut 5 (*1/10 s) - par défaut 5.
Face de dé	- nom du fichier contenant le dessin de la face - action liée à la face	- chaîne de caractère, par défaut vide. - action (voir plus haut)

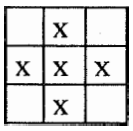
6.3. Commentaire

- Pour chaque jeton, le concepteur devra définir le mode de manipulation. Par défaut, lorsqu'un jeton est créé, son mode de manipulation est déplacement, il n'est pas superposable et n'a aucune caractéristique. Le nombre de sélections avant de contrôler la validation des sélections est de deux, le temps d'affichage des jetons sélectionnés, lorsque l'on a sélectionné le dernier jeton, est de : 5 * 1/10 de secondes. Pour les types de déplacements : dans la grille tous les déplacements sont autorisés ainsi qu'entre tous les échiquiers (grille, sabot, zone d'attente)..

- Pour les règles d'association, lorsque la validation par caractéristique est validée, par défaut le mouvement (sélection) est accepté. Si la validation est acceptée, cela signifie qu'il y aura incrémentation du nombre de déplacements mais pas du nombre d'erreurs. Pour la

sélection, il n'y a pas de différence, lorsque la sélection n'est pas validée, entre accepter la sélection et refuser la sélection. Dans tous les cas, il y aura incrémentation du nombre d'erreurs et les jetons seront retournés, éventuellement après un certain temps.

- Par défaut, la grille est mélangée avant toute nouvelle partie. Les jetons déclarés immobiles restent à la place définie par le concepteur. Les déplacements d'une seule case se font en utilisant la table des déplacements de la case d'origine du déplacement. Si cette case n'a pas de table de déplacement, c'est la table de déplacement de la grille qui est utilisée. Par défaut, la table de déplacement des cases de la grille est définie :



Les déplacements de x cases, se font par vol et n'utilisent pas la table de déplacement.

6.4. Définition des règles du jeu

Pour définir les règles d'un jeu, le concepteur doit donc :

- déterminer le type de manipulation permis sur les jetons,

- associer une liste de caractéristiques aux jetons (optionnel),

- choisir entre pas de validation, validation par caractéristique maximum et validation par caractéristique minimum (optionnel).

Voyons comment, à l'aide de ces concepts, nous pouvons définir les règles des jeux analysés.

6.5. Le formulaire

Tout concept qui peut être défini de manière graphique ou dont la prise en compte est liée à un concept défini de manière graphique ne sera pas repris dans le formulaire. L'approche graphique est retenue car elle est plus intuitive. Par exemple, les coordonnées de la grille indique à quel endroit il faut commencer à dessiner la grille. Ces coordonnées peuvent être cachées du point de vue du concepteur si on lui permet de déplacer la grille à l'aide de la souris. De cette manière il ne sera pas confronté à cette caractéristique de la grille.

La grille, le sabot, la zone d'attente, la zone de résultat et la case peuvent être définis de manière graphique et sont donc retirés du formulaire de définition des règles du jeu.

Le concept de zone est lié à celui de grille, l'interface de la grille sera donc liée à l'interface des zones de la grille. Une zone est liée à deux concepts : le concept de case et le concept de caractéristiques.

Le concept de joueur peut également être géré par l'interface car il est lié aux différents échiquiers. Si le concepteur a défini plusieurs joueurs pour un jeu, chaque joueur peut disposer de son propre échiquier, de sa propre zone de résultat. Si l'on peut définir de manière graphique les liens existants entre les joueurs et les sabots, les zones d'attente et les zones de résultat ; la caractéristique identifiant le sabot, la zone de résultat et la zone d'attente devient inutile. Le lien existant entre les échiquiers et les joueurs sera géré par l'interface.

De même, le concept de tour de jeu est tellement lié au concept de joueur, que l'on pourra soulager le formulaire des règles du jeu en l'incluant dans l'interface du joueur.

Les deux derniers concepts pouvant être définis graphiquement sont le dé et la face de dé.

Formulaire	Caractéristiques
Règles d'association	- validation des caractéristiques - action à effectuer lorsque les caractéristiques sont validées - action à effectuer lorsque les caractéristiques ne sont pas validées.
Jeu Gagné lorsque	- la grille est vide - la grille est remplie - le sabot est vide - toutes les zones sont valides - zones valides et sabot vide - zones valides et grille remplie - afficher boîte de dialogue gagné - message gagné
Jeu Perdu lorsque	- nombre max. de coups atteint - nombre max. erreur atteint - afficher boîte de dialogue perdu - message perdu

6.6. Retour aux jeux

Bien que le formulaire créé puisse paraître simpliste, montrons qu'à l'aide de ce formulaire nous pouvons créer plusieurs jeux parmi les jeux étudiés. Dans les tableaux qui suivent, je n'ai repris que les éléments nécessaires à la définition de ces jeux.

Le jeu de mémoire

Concepts	Caractéristiques	Valeurs
Règles d'association	- validation des caractéristiques - action si validation	- validation maximum - enlever les jetons
Jeu Gagné lorsque	- la grille est vide	- oui
Jeu Perdu lorsque	- nombre max. de coups atteint - nombre max. erreur atteint	- oui ou non - oui ou non
Jeton	- caractéristiques - superposable - mode de manipulation	- liste de caractéristiques - non - sélection
Grille	- ordre jetons de la grille	- ordre aléatoire
Sabot	- présent	- non
Zone d'Attente	- présente	- non
Zone de Résultat	- présente - remplissage - zone remplie	- oui - par ligne - décaler ligne

Commentaire, bien que la zone de Résultat ne soit pas un élément nécessaire à la définition des jeux, elle a été utilisée.

Le jeu de taquet

Concepts	Caractéristiques	Valeurs
Règles d'association	- validation des caractéristiques - action si non validation	- validation minimum - accepter le déplacement
Jeu Gagné lorsque	- toutes les zones sont valides	- oui
Jeton	- caractéristiques - superposable - mode de manipulation - type de déplacement - dans la grille	- une caractéristique unique par jeton - non - déplacement - une case
Grille	- zones - ordre jetons de la grille	- autant de zones que de cases dans la grille - ordre aléatoire
Zones	- caractéristiques « concepteur »	- la caractéristique du jeton devant se trouver dans cette zone (case)
Sabot	- présent	- non
Zone de Résultat	- présente	- non
Zone d'Attente	- présente	- non

Commentaire : Pour plus de facilité, la table de déplacements par défaut de la grille a été utilisée. Elle convient bien pour ce jeu. Il faut cependant changer les tables de déplacements de toutes les cases se trouvant en bord d'échiquier de manière à interdire à un jeton se trouvant en bord d'échiquier de le traverser.

Le jeu des familles

Concepts	Caractéristiques	Valeurs
Règles d'association	- validation des caractéristiques - action en cas de validation du déplacement ou de la sélection - action en cas de non validation du déplacement	- validation minimum ou maximum - accepter - refuser le déplacement
Jeu Gagné lorsque	- zones valides - message gagné	- oui « Vous avez réussi »
Jeton	- caractéristiques - mode de manipulation - type de déplacement - sabot vers grille - grille vers sabot - grille vers z attente - sabot vers z attente - dans la grille	- liste de caractéristiques - déplacement - oui - oui - oui ou non - oui ou non - tout permis
Grille	- zones	- liste de zones
Zones	- cases appartenant à la zone - caractéristiques « concepteur »	- liste de cases - liste de caractéristiques
Sabot	- présent	- oui
File d'attente Sabot	- nombre de cases - ordre jetons	- nombre de jetons à placer - ordre définit ou aléatoire
Zone de Résultat	- présente - remplissage - zone remplie	- oui ou non - par ligne ou par colonne - vider zone
Zone d'Attente	- présente	- oui ou non

Commentaire : Il y a plusieurs variantes possibles. Le concepteur peut fixer les caractéristiques d'une zone. Pour cette variante, il va choisir d'enlever les jetons validés, d'effectuer une validation maximum des caractéristiques et d'accepter les déplacements lorsqu'il n'y a pas de validation des jetons. Le jeu est fini lorsque toutes les zones de la grille sont validées et lorsque le sabot (et la zone d'attente) est (sont) vide. Dès que les caractéristiques d'un jeton sont validées, le jeton est enlevé de la grille.

Le concepteur peut ne pas fixer de caractéristiques aux zones de la grille. Il laisse le joueur trouver lui-même les points communs entre les jetons, il va donner plusieurs caractéristiques aux jetons et demander une validation par caractéristique minimum. Lorsqu'un déplacement n'est pas validé, il est refusé. Il n'y a pas de zone de résultat.

Le jeu de l'oie

Concepts	Caractéristiques	Valeurs
Règles d'association	- validation des caractéristiques - action en cas de validation du déplacement ou de la sélection - action en cas de non validation du déplacement	- validation maximum - mettre joueur en sommeil -accepter ou refuser le déplacement
Jeton	- caractéristiques - superposable - mode de manipulation - type de déplacement - dans la grille	- caractéristiques se trouvant dans les jetons. - oui - déplacement - à l'aide d'un dé
Joueur	- nom	- chaînte de caractère
Dé	- faces	- liste de faces
Face de dé	- action liée à la face	- voir plus haut
Jeu Gagné lorsque	- toutes les zones sont valides	- oui ou non
Grille	- zones	- liste de zones
Case de grille	- numéro	- nombre
Zones	- cases appartenant à la zone - caractéristiques « concepteur »	- liste de cases - liste de caractéristiques
Sabot	- présent	- oui

Remarque : Le concepteur définit deux zones :
La première, est une zone composée d'une seule case : la case d'arrivée du jeu de l'oie. Le concepteur définit pour cette zone une caractéristique. Pour les jetons créés, le concepteur définit la même caractéristique. Le sabot est composé d'une seule case, c'est la case du début du jeu de l'oie. Le concepteur crée plusieurs dés. Il y a un dé qui est utilisé pour déplacer les jetons et les autres qui sont placés dans les cases de la grille. Lorsqu'un dé fait retourner un jeton vers le sabot, le jeton est dirigé vers la première case de la grille. Lorsqu'un jeton se présente dans la case d'arrivée, tous les jetons du joueurs sont validés, le joueur est mis en sommeil. Lorsque tous les joueurs sont mis en sommeil, le jeu s'arrête.
La deuxième zone est le reste de l'échiquier.

Le concepteur du jeu détermine à l'avance le nombre de joueurs. Il faut en effet autant de jetons qu'il y a de joueurs. Dans chaque jeton, le concepteur y ajoute une caractéristique spéciale identifiant le joueur. Il n'y a qu'un seul sabot utilisé par tous les joueurs.

7. Conception du logiciel

7.1. Ensemble minimal

Avant d'entamer cette nouvelle partie, j'ai été contraint de laisser tomber le concept de dé. Celui-ci risquait d'être complexe, il était préférable de mettre de côté ce concept pour éviter de compromettre l'entièreté du travail.

Je ne ferai donc plus mention de ce concept dans le reste du travail. Les concepts de jeton, sabot, règles d'association, règles de fin de jeu sont déjà suffisants pour envisager un grand nombre de jeux. De plus le logiciel réalisé étant suffisamment modulaire, il sera possible de rajouter ce concept sans trop de modifications dans le reste du logiciel.

Pour rajouter ce concept, il suffira juste de changer la partie règles du jeu qui traite du déplacement des jetons, ainsi que le message « Déplacement » envoyé à une case. Pour traiter les dés liés à un jeton et les dés liés à une case, il faut changer le message "Déplacement" vers une case.

A partir de maintenant, je vais arrêter de faire référence au travail effectué par Yves Lowette et Jean-Christophe Marchal. Leurs idées sur l'interface et l'implémentation du logiciel sont éloignées de ce que je propose.

Mon idée est de proposer un comportement par défaut pour tous les concepts utilisés. Si un concept n'est pas défini par le concepteur, cela ne signifie pas qu'il ne sera pas présent. Cette idée permettra au concepteur de développer le plus rapidement possible un jeu. De plus, cette approche permet une construction interactive du jeu. Le concepteur définit un certain nombre de concepts en gardant les options par défaut pour les autres. Le jeu créé peut être testé. Le concepteur a ainsi un feed-back immédiat des modifications apportées aux différents concepts.

7.2. Choix du langage

Le logiciel manipule des objets graphiques (jeton, grille, ...) et doit pouvoir être utilisé avec la souris. Il est clair que le générateur doit tourner sous Windows.

L'éventail des langages de programmation sous Windows est large. Une autre caractéristique du logiciel est son caractère modulable. Pour cette raison, le choix d'un langage objet est préférable.

Parmi les langages à objets tournant sous Windows, j'ai opté pour le Visual C++. Ce choix est personnel, c'est un langage que je désirais apprendre et j'ai saisi l'occasion de ce mémoire. Une des raisons particulières de ce choix est sa présence sous de nombreuses plates-formes de développement : Windows, Windows NT et Mac OS. Une seconde raison est qu'il

semble devenir incontournable (même Borland a adopté les MFC dans sa version 5.0 du Borland C++).

Le logiciel créé est modulable. Je n'ai pas implémenté le concept de dé. Montrons que ce concept peut aisément être rajouté au logiciel. Supposons que l'on crée un objet dé, une interface liée à cet objet ainsi que des fonctions de traitement des messages envoyés à un dé, les modifications à apporter au reste du programme sont minimes .

Pour un dé lié à un jeton, il faut modifier le comportement du jeton lorsqu'il est sélectionné. Lorsqu'un jeton est sélectionné, il reçoit un message de type Sélection. Un jeton dont le déplacement se fait par dé recevant un message de type sélection doit envoyer un message "Sélection " au dé, provoquant l'affichage du dé. Lorsque le joueur clique sur le dé, le dé est lancé et l'action liée à la face du dé est exécutée. Si l'action est de type déplacement du dé (vers la zone de résultat, vers le sabot, à l'intérieur de la grille), le dé va envoyer un message de type déplacement au jeton. Si l'action est de type « passer son tour », le dé va envoyer ce message au propriétaire du jeton ...

Lorsqu'un jeton se déplace dans une case, la case reçoit un message "Déplacement", avec, entre autres, comme paramètre, un pointeur sur le jeton effectuant le déplacement. Dans, ce cas, il suffit de lancer le dé, une fois le dé lancé, celui-ci envoie un message au jeton pointé avec comme paramètre l'action à effectuer ...

7.3. Nouveaux concepts

Comme je l'ai signalé lors de l'analyse, certains jetons sont superposables. Cela signifie que plusieurs jetons peuvent se poser dans une même case. Pour gérer le concept de jetons superposables, nous considérons qu'il n'y a qu'un seul jeton visible à un moment donné dans une case. Ce jeton est appelé le jeton visible. Le joueur peut à tout moment, demander l'affichage du jeton suivant.

Le concept de jeton visible d'une case va grandement faciliter la conception du logiciel. Chaque fois qu'un message sera envoyé vers une case, ce message sera envoyé si nécessaire vers le jeton visible, sans devoir passer par une procédure de demande de jeton.

7.4. L'identification des objets

D'après l'analyse des jeux on peut identifier les objets suivants : "la grille", "le sabot", "la file d'attente du sabot", "la zone d'attente", "la zone de résultat", "la zone de grille", "le joueur", "la case" et "le jeton".

Etant donné la grande similitude existante entre les objets : "grille", "sabot", "zone d'attente" et "zone de résultat", je vais créer l'objet abstrait "échiquier". Tous ces objets vont avoir l'objet "échiquier" comme classe de base. Parmi les échiquiers, la grille est unique, tandis que les autres sont optionnels mais peuvent être présents plusieurs fois.

De manière à éviter la gestion d'objets optionnels, je vais créer les objets liste de sabot, liste de zone d'attente et liste de zone de résultat. Ces objets sont également uniques. De nouveau on peut observer une grande similitude entre ces objets, c'est pourquoi je vais créer l'objet liste échiquier.

La grille pouvant être découpée en une ou plusieurs zones, il convient de créer une classe zone. Cette classe est obligatoire. Lors de la création de la grille, une zone "Reste Grille" est créée.

Chaque échiquier possède une ou plusieurs cases. D'après l'analyse, on constate que les cases de la grille ont une particularité : elles possèdent une table de déplacement. Il y a donc création d'une classe "case de grille" ayant la classe "case" comme classe de base".

La classe jeton est particulière. On peut imaginer plusieurs modélisations possibles. Lorsque le concepteur demande 10 jetons "Boule verte", on peut :

- créer et manipuler 10 jetons "Boule verte"
- manipuler 10 pointeurs sur l'objet "Boule verte".

J'ai longtemps pensé utiliser des pointeurs sur un objet. Etant donné qu'il n'y a qu'un seul jeton et plusieurs pointeurs vers ce jeton, il n'est pas possible de sélectionner un de ces jetons. La solution consiste à sélectionner la case. Le jeton sélectionné étant le jeton visible de la case. Lorsqu'une case est sélectionnée, elle ne peut plus changer de jeton visible, ce qui entraîne une restriction. Une deuxième restriction est qu'étant donné qu'il est impossible de différencier les jetons créés, il n'est pas possible d'attribuer dynamiquement un jeton à un joueur.

Pour ces deux objections, j'ai créé une classe liste jetons. Cette classe contient les jetons types. Un jeton type est un jeton créé par le concepteur, une des caractéristiques est le nombre de jetons de ce type pouvant être créé. Chaque fois qu'un jeton sera utilisé, soit placé dans la grille, soit dans un sabot, un jeton sera créé.

La classe jeton contient un pointeur sur le type de jeton contenant ses caractéristiques, une liste de pointeurs sur les joueurs propriétaires et un pointeur vers la case dans laquelle il se trouve.

La classe règles du jeu. Le choix de représenter les règles du jeu par un objet unique peut sembler étrange. Ce choix a été fait en fonction de l'optique de modularité. Il permet par exemple, lors d'une version future, d'implémenter le concept de variante. Une petite modification dans les règles du jeu peut parfois entraîner de grands changements dans la manière de jouer, c'est le concept de variante (non implémenté). Avec le concept de variante, on pourrait créer plusieurs objets règles du jeu, chaque objet correspondant à une variante différente du même jeu. L'interface du jeu proposerait le choix au joueur entre les différentes variantes.

La classe "joueur". Par similitude avec les classes "liste échiquier" et "liste jeton", j'ai créé une classe liste joueur. Cette classe a pour but principal de garder un pointeur sur le joueur courant "ayant la main".

La classe caractéristiques. Cette classe n'est pas indispensable. Elle gère un tableau de caractéristiques, c'est à dire un tableau de strings. Son but est de convertir :

- un entier en string, lors d'une demande de caractéristique
- un string en entier, lors de l'introduction d'une nouvelle caractéristique

Cette classe est très peu utilisée car toutes les manipulations sur des caractéristiques à l'intérieur de logiciel se font par des manipulations sur des entiers. Elle est utilisée dans les fonctions d'interface avec l'utilisateur, le concepteur voit les caractéristiques comme des chaines de caractères.

Le fait d'utiliser un objet pour représenter un tableau unique permet une indépendance dans la représentation du tableau.

La classe ECHIQUIER

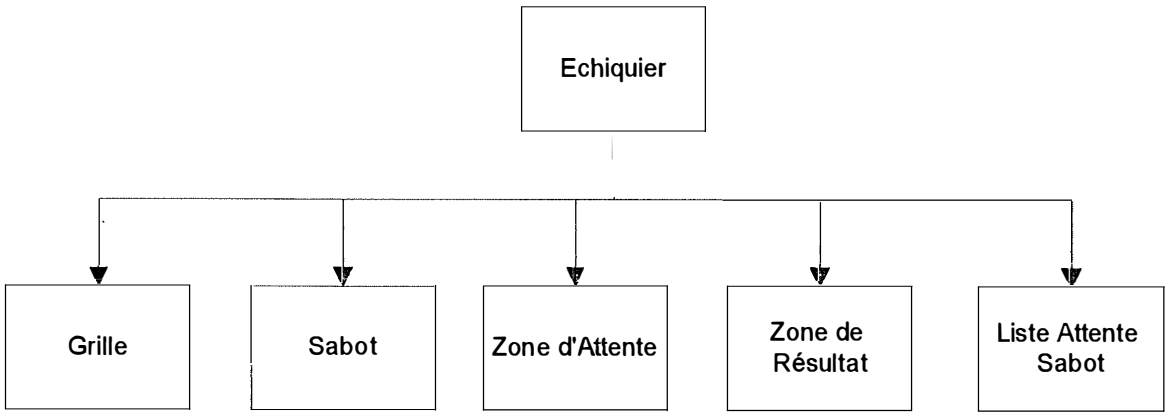
Un échiquier est composé de :

- nombre de lignes : integer
- nombre de colonnes: integer
- hauteur des lignes : integer
- hauteur des colonnes : integer
- espace entre les lignes : integer

- espace entre les colonnes : integer

- une liste de pointeurs sur des objets case de l'échiquier

La grille, le sabot, la zone d'attente et la zone de résultat sont des échiquiers. La file d'attente du sabot peut être vue comme un échiquier. Il s'agit d'un échiquier composé d'une seule ligne dont la hauteur et la largeur des cases sont celles définies dans le sabot et dont l'espace entre les lignes et entre les colonnes est de 0. Le fait de dériver la file d'attente de l'objet échiquier va permettre à la file d'attente d'utiliser toutes les fonctions définies pour l'objet échiquier.



La classe GRILLE

La grille est un échiquier composé de :

- table de déplacement. Il s'agit de 9 variables BOOL, indiquant pour chacune des 8 directions, si un déplacement dans cette direction est possible. La case du centre indique si cette case est éligible pour un déplacement.

- une liste de pointeur sur des objets zone.

Il existe deux occurrences de cette classe : la grille concepteur et la grille joueur. Ces deux occurrences sont obligatoires. La grille concepteur est définie par le concepteur au moment de la création du jeu. La grille joueur est créée à chaque nouvelle partie à partir des options définies par le concepteur. Soit les jetons de la grille concepteur sont placés aux mêmes endroits dans la grille joueur, soit ils sont placés de manière aléatoire (sauf pour les jetons déclarés immobiles).

La classe ZONE

La classe zone est composée :

- une liste de pointeurs sur des objets case de la grille

- caractéristiques « concepteurs ». Il s'agit d'un tableau de numéros de caractéristiques (tableau integer) . Une caractéristique concepteur est une caractéristique que le concepteur a définie pour la zone.

- caractéristiques de la zone. Un tableau d'entiers. Les caractéristiques de la zone sont la réunion entre les caractéristiques concepteurs et les caractéristiques des jetons se trouvant dans la zone.

- nom de la zone : string.

La classe SABOT

Un sabot est un échiquier regroupant les jetons qui seront manipulés par le joueur. Un sabot est composé de deux parties, la partie visible, c'est l'échiquier affiché à l'écran et la partie cachée, c'est l'ensemble des jetons se trouvant dans la partie cachée du sabot ou la file d'attente du sabot.

Un sabot est composé de :

- un pointeur sur deux objets file d'attente. La première file d'attente est la file d'attente concepteur (les jetons se trouvant dans cette file d'attente sont placés par le concepteur du jeu). La deuxième file d'attente est la file d'attente joueur ou jeu, il s'agit de celle qui sera utilisée pour le jeu. Lors de chaque nouveau jeu, il y a copie des jetons se trouvant dans la file d'attente concepteur vers la file d'attente joueur (pour chaque sabot). La copie se fait en gardant l'ordre de la file d'attente concepteur ou de manière aléatoire, selon le choix du concepteur.

- une liste de pointeurs sur le(s) joueur(s) propriétaire(s) du sabot

La classe ZONE DE RESULTAT

Une zone de résultat est un échiquier composé de :

- option de remplissage une des 4 strings suivantes : "decaler_ligne", "decaler_colonne", "vide", "rien"

- une liste de pointeur sur le(s) joueur(s) propriétaire(s) de la zone de résultat

- remplissage par ligne : BOOL (par ligne si vrai, par colonne si faux).

La classe ZONE D'ATTENTE

Une zone d'attente est un échiquier composé de :

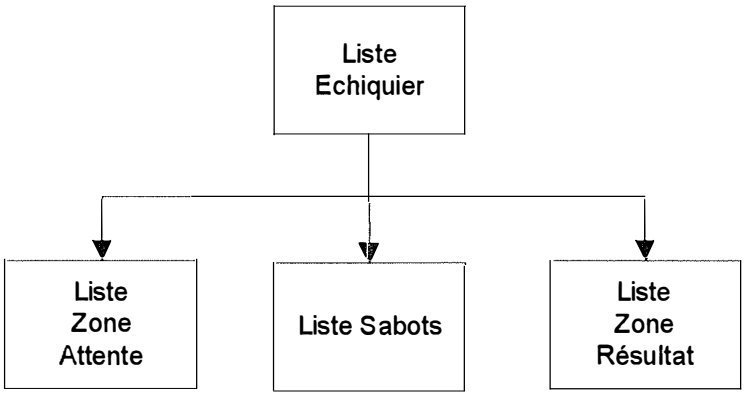
- une liste de pointeur sur le(s) joueur(s) propriétaire(s) de la zone d'attente

La classe LISTE ECHIQUIER

Un objet de type liste échiquier est un objet gérant une liste de pointeurs sur des échiquiers. Elle est composée de :

- liste de pointeurs sur des objets échiquier

La classe LISTE SABOTS



Une liste de sabots est une liste d'échiquiers composée de :

- un string représentant l'option de remplissage pour chacun des sabots. Ce string peut avoir les valeurs suivantes : "quitte_sabot" indique que chaque fois qu'un jeton quitte le sabot il faut le remplacer par un jeton de la file d'attente, "sabot_vide" indique qu'il ne faut remplir le sabot que lorsque le sabot est vide, "demande" le sabot est rempli à la demande du joueur.

Il m'a semblé utile de stocker l'option de remplissage des sabots dans la liste des sabots. Cela signifie que tous les sabots auront la même option de remplissage, le concepteur ne devant changer cette option qu'une seule fois pour tous les joueurs. Permettre des options de remplissages différentes pour les joueurs ne m'a pas semblé "loyal" pour le joueur dont le sabot ne se remplit que lorsqu'il est vide alors que celui des autres joueurs se remplit à la demande.

La classe LISTE ZONES RESULTAT

Une liste de zones de résultat est une liste d'échiquiers. Contrairement à la liste de sabots, les options de remplissages des zones de résultat sont stockées dans chaque zone de résultat. Cela demandera plus de travail au concepteur mais il me semble que pour des raisons esthétiques, le concepteur peut choisir un remplissage par ligne pour une zone de résultat et un remplissage par colonne pour une autre.

La classe LISTE ZONES ATTENTE

Une liste de zones d'attente est une liste d'échiquiers.

La classe CASE

Une case est un composant d'un échiquier. Chaque échiquier gère une liste de pointeurs sur ses cases. Chaque case, garde un pointeur sur l'échiquier dont elle fait partie. Une case est composée de :

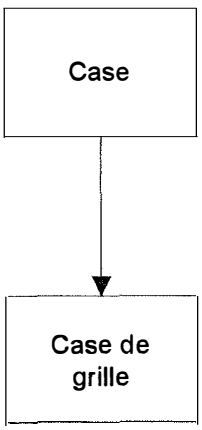
- le numéro de la ligne dans laquelle se trouve la case : entier.

- le numéro de la colonne dans laquelle se trouve la case : entier.

- un pointeur sur l'échiquier propriétaire de la case : pointeur sur échiquier .

- une liste de pointeurs sur les jetons se trouvant dans la case.

Un case d'un échiquier est identifiée par son numéro de ligne et son numéro de colonne.



La classe CASE DE GRILLE

Une case de grille est une case composée de :

- une table des déplacements possibles : 9 BOOL

- un pointeur vers la zone dans laquelle se trouve la case

La classe LISTE JOUEURS

Il n'existe qu'un seul objet de la classe liste joueurs composé de :

- une liste de pointeurs sur des objets joueur

- un pointeur sur le joueur courant (joueur qui a la main, joueur en cours de modification, ...)

La classe JOUEUR

Un objet joueur, est composé de :

- un nom : string.

- un pointeur sur son sabot.

- un pointeur sur sa liste d'attente.

- un pointeur sur sa zone de résultat.

La classe LISTE JETONS

La classe contient :

- une liste de pointeurs (statique) sur des jetons types

- un pointeur (statique) sur le jeton type courant.

- la taille des jetons (statique) : deux integer (hauteur et largeur).

- jeton type, composé de :

  - le chemin vers le BITMAP utilisé pour dessiner le jeton : un string.

- un tableau d'entiers pointant sur des entrées dans le tableau des caractéristiques.

- le déplacement dans la grille : un string pouvant avoir les valeurs suivantes : immobile_grille, tout_permis, une_case, x_cases

- les déplacements possibles, une série de BOOL indiquant si les déplacements suivants sont autorisés : grille vers sabot, grille vers attente, sabot vers attente, attente vers grille et attente vers sabot.

- le mode de sélection du jeton, un string pouvant avoir les valeurs suivantes : déplacement, immobile, sélection.

- le nombre de jetons de ce type pouvant être créé

- une liste de pointeurs sur les jetons de ce type

La classe Jeton

Un objet de la classe jeton est composé de :

- un pointeur de la case dans laquelle il se trouve

- un pointeur sur un objet type

- un bool indiquant si le jeton est sélectionné

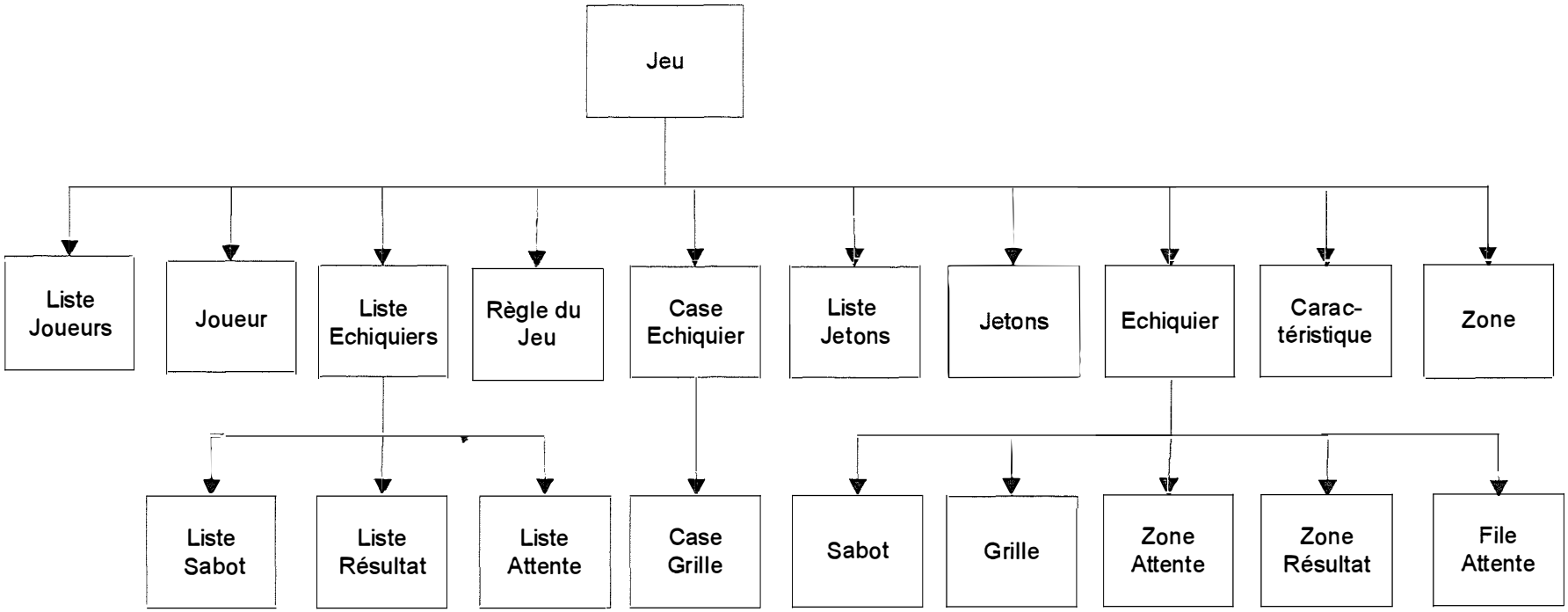
La classe REGLES DU JEU

Il n'existe qu'une seule occurrence de la classe règles du jeu. L'objet règles du jeu stocke les choix effectués par le concepteur lors de la conception d'un jeu (validation par caractéristique minimum ou maximum, action en cas de validation : accepter ou enlever jetons, action en cas de non validation: accepter le déplacement ou refuser, règles de fin de jeu, ...) et contrôle la validation des déplacements lors d'un jeu. Pour valider dynamiquement les règles, la classe règles du jeu tient à jour une liste de pointeurs sur les jetons sélectionnés.

La classe CARACTERISTIQUES

Il n'existe qu'une seule occurrence de la classe caractéristique. L'objet caractéristiques gère un tableau de caractéristiques (tableau de strings).

L'architecture



8. Validation d'un déplacement

Le générateur est composé de deux sous-ensembles :

- la conception d'un jeu
- la partie jeu, permettant à un joueur de jouer avec un jeu défini dans la partie de conception.

La partie conception d'un jeu est composée d'un ensemble d'interfaces offrant différents services au concepteur. L'interface sera présenté en détail lors d'un prochain chapitre.

La partie jeu est composée de deux algorithmes : l'algorithme d'initialisation du jeu et l'algorithme de validation des déplacements.

Lors de l'initialisation d'un jeu, il y a copie des objets créés par le concepteur en objets jeu. Il y a création d'une grille_jeu, d'un (ou plusieurs) sabot_jeu (avec les files d'attente_jeu liées). Pour chaque jeton se trouvant dans la grille concepteur ou dans un sabot concepteur, il y a création d'un jeton jeu qui sera posé dans la grille jeu ou dans le sabot jeu. La création d'objets jeu a pour but de garder intacte la définition du jeu faite par le concepteur.

Lorsque l'initialisation d'un jeu se termine, le programme entre en boucle de validation des déplacements jusqu'à ce qu'une condition entraînant la fin du jeu soit rencontrée.

L'algorithme réagit lorsque le joueur clique dans une case. La première partie de l'algorithme va être d'interpréter l'action désirée par le joueur et la seconde partie va être de contrôler cette action.

Interprétation

Lorsque le joueur clique dans une case, quelle action désire-t-il faire ? une sélection d'un jeton se trouvant dans cette case ou un déplacement d'un jeton vers cette case ?

La sélection

Il y a sélection d'un jeton lorsque :

- aucun jeton n'est sélectionné.

- le joueur choisit une case. Le jeton sélectionné est le jeton visible de cette case.

ou

- au moins un jeton dont le mode de manipulation est "sélection" est sélectionné.

- l'utilisateur sélectionne une case différente de celle contenant un jeton sélectionné. Si la case choisie contient un jeton sélectionné, ce jeton est désélectionné si cette action est autorisée par le concepteur.

Le déplacement

Il y a déplacement lorsque :

- un jeton dont le mode de manipulation est "déplacement" est sélectionné.

- l'utilisateur choisit une case.

Contrôles effectués lors d'une sélection d'un jeton

Contrôle de la case choisie

La case C choisie sera validée en fonction du mode d'action effectué par le joueur (sélection ou déplacement), en fonction de l'état de la case (la case peut-elle recevoir des jetons ou non) et en fonction des jetons se trouvant dans la case (y a-t-il un jeton non superposable dans cette case ou non)..

	sélection	déplacement
C n'est pas une case de jeu (d'après la table de déplacement de la case)	ko	ko
C est vide ou composée uniquement de jetons immobiles	ko	ok
un jeton non superposable se trouve dans C	ok	ko
tous les jetons de C sont superposables	ok	ok

Contrôle du jeton visible

Si le contrôle de la case est validé, le jeton visible J sera sélectionné si :

Mode de manipulation de J	résultat du contrôle
immobile	ko, un jeton immobile ne peut pas être manipulé par le joueur
sélection	ok
déplacement	ok si aucun jeton n'est sélectionné, sinon ko

Si la sélection est validée, il y a contrôle du mode de manipulation du jeton. Si ce mode est sélection, il y a contrôle du nombre de sélections déjà effectuées. Si ce nombre est atteint, il y a contrôle des caractéristiques des jetons sélectionnés.

Contrôle du déplacement

Ce contrôle se fait à trois niveaux :

- niveau du jeton : le déplacement effectué par le joueur doit correspondre avec le type de déplacement du jeton. Le type de déplacement dans la grille est : tout permis, déplacement d'une case, de x cases, immobile. En plus du mode de manipulation, le concepteur choisit d'accepter ou de refuser les déplacements suivants : de la grille vers le sabot, de la grille vers la zone d'attente, du sabot vers la zone d'attente, de la zone d'attente vers le sabot.

- niveau des règles du jeu : Il s'agit d'offrir au concepteur un moyen de permettre ou de refuser des déplacements vers la grille ou à l'intérieur de la grille à l'aide du contrôle de validation par caractéristiques. Ce contrôle est effectué uniquement si le jeton que l'on désire déplacer possède au moins une caractéristique.

- niveau de la grille :

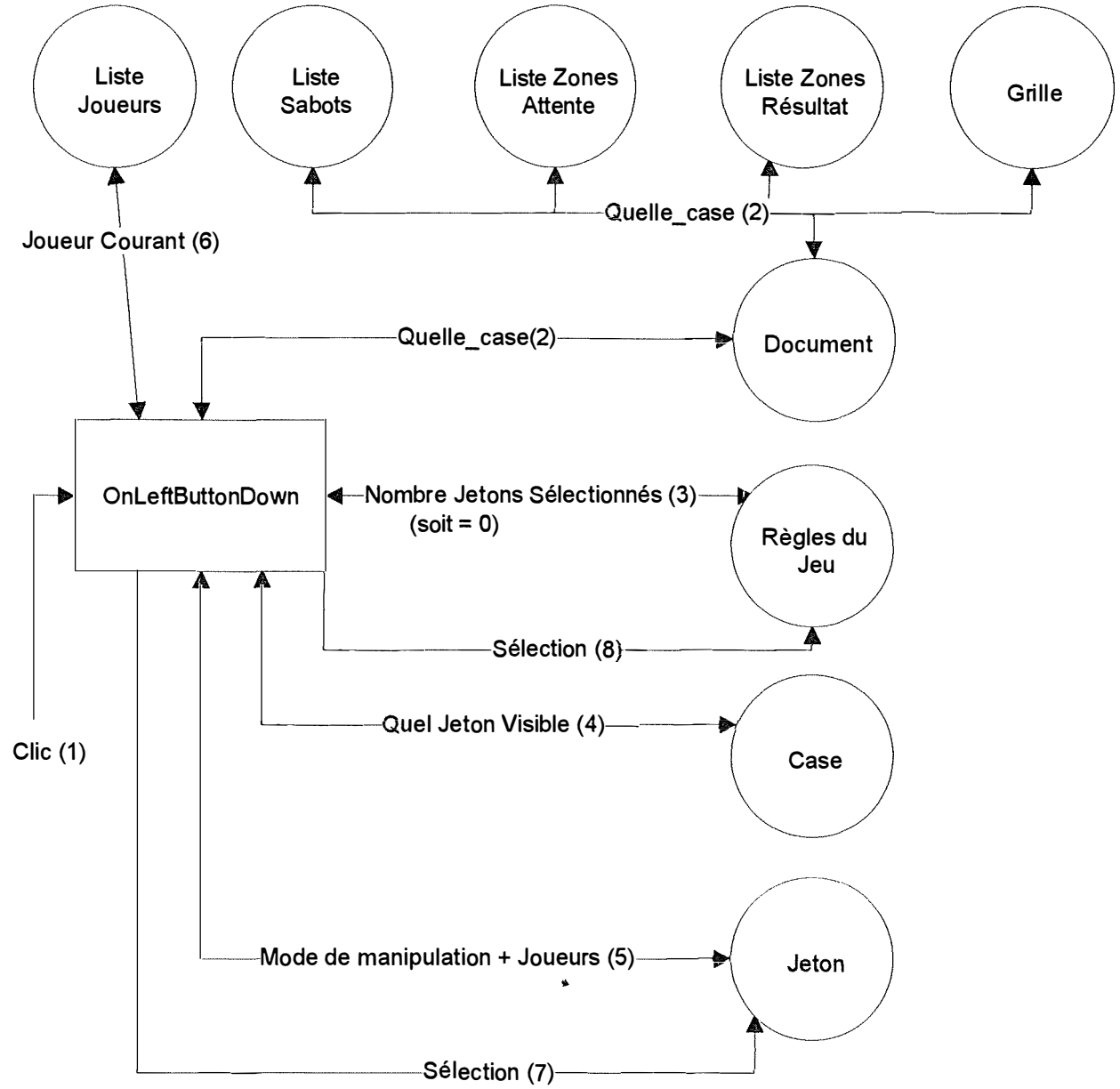
    - si la case de destination du déplacement est une case de la grille, il doit s'agir d'une case de jeu, c'est-à-dire d'une case ayant une croix au centre de sa table de déplacement.

    - si les cases d'origine et de destination du déplacement sont des cases de la grille et si le type de déplacement du jeton sélectionné est "déplacement d'une case", le déplacement effectué doit correspondre avec la table de déplacements de la case

    - si les cases d'origine et de destination du déplacement sont des cases de la grille et si le type de déplacement du jeton sélectionné est "déplacement de x cases", la case de destination ne peut pas se trouver à plus de x lignes ou plus de x colonnes de la case d'origine du déplacement.

8.2. Messages générés lorsqu'un joueur clique dans une case

lorsque aucun jeton n'est sélectionné



(1) Clic : Lorsqu'un joueur appuie sur le bouton gauche de la souris, l'interface génère le message "LeftButtonDown" (que nous appellerons clic). La fonction "OnLeftButtonDown" est activée en réponse à ce message. Le paramètre de cette fonction est de type CPoint. Les variables membres d'un objet de type Cpoint sont : int cx et int cy, où cx et cy sont des coordonnées écran.

(2) Demande à la classe Document, si les coordonnées du point ne se trouvent pas à l'intérieur d'une case. La classe Document demande aux classes Grille, Liste Sabots, Liste Zones

Attente, Liste Zones Résultat, quelle classe contient le point. Si aucune classe ne contient ce point, la classe Document transmet NULL, sinon elle transmet UN pointeur sur un objet case (parmi tous les points contenant cette case). Les différents échiquiers sont supposés ne pas se superposer.

(3) Demande à la classe Règle du Jeu, le nombre de jetons actuellement sélectionnés. La classe règle du jeu garde une liste de pointeurs sur les objets sélectionnés. Supposons dans ce cas que ce nombre est de zéro. Dans ce cas, on interprète le clic de la souris dans une case comme une sélection du jeton visible de cette case.

(4) Demande à la case, un pointeur sur son jeton visible. S'il n'y a pas de jetons dans la case, il ne se passe rien (sortie de la fonction OnLeftButtonDown).

(5) Demande au jeton son mode de manipulation et une liste de pointeurs sur le(s) joueur(s) propriétaire(s) du jeton. Si ce mode est "immobile", il n'y a pas de sélection et la fonction OnLeftButtonDown se termine.

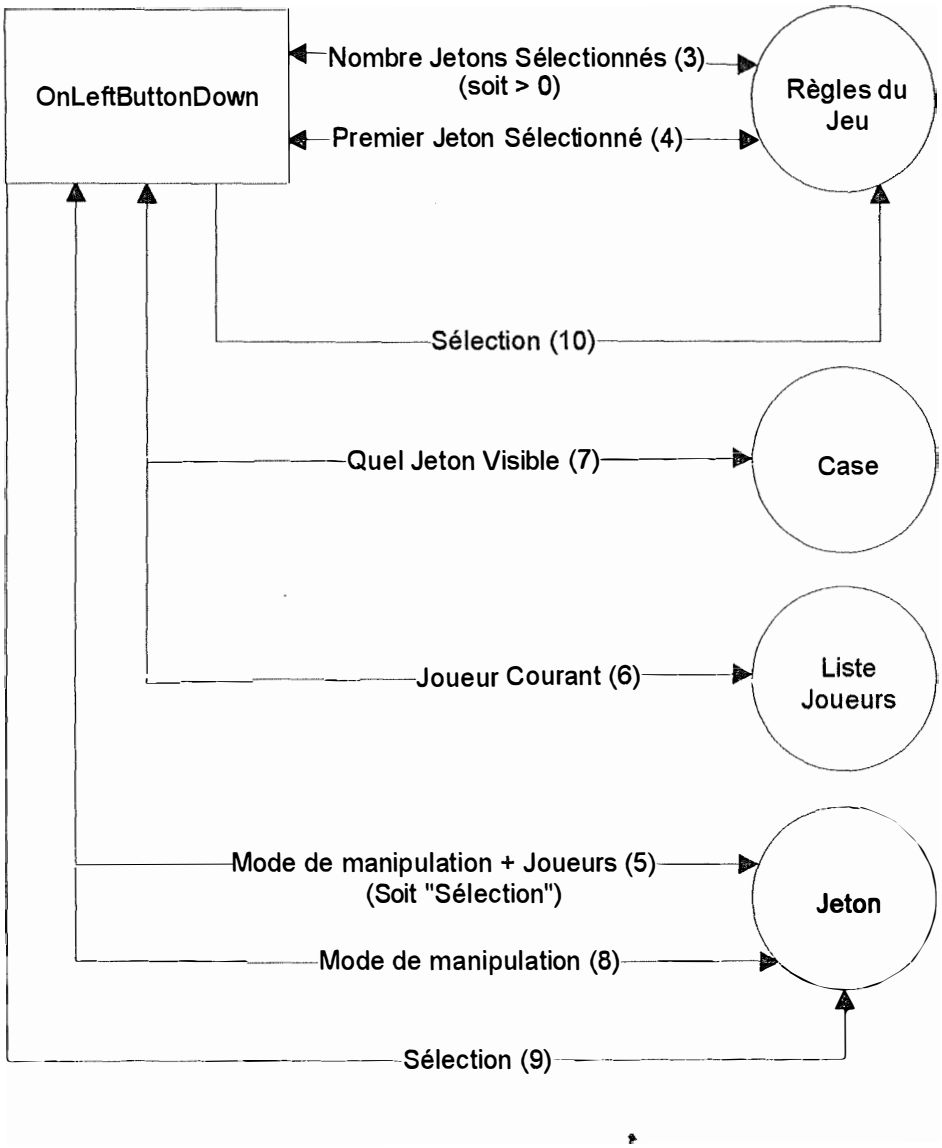
(6) S'il y a au moins un joueur propriétaire du jeton, le joueur courant doit être propriétaire du jeton. Si ce n'est pas le cas, fin de OnLeftButtonDown et pas de sélection.

(7) Si le mode de manipulation est "sélection" ou "déplacement", un message "Sélection" est envoyé au jeton. Le jeton garde son état. Lorsqu'il est sélectionné, il s'affiche en vidéo inverse.

(8) Un message "sélection" est envoyé à l'objet règle du jeu avec comme paramètre un pointeur sur le jeton sélectionné. L'objet règle du jeu doit ajouter ce pointeur à sa liste de pointeurs sur les objets sélectionnés.

Lorsque au moins un jeton est sélectionné .

b1) soit le mode de manipulation du jeton sélectionné est de type "sélection"



(1) et (2) sont les mêmes que au point (a), message clic envoyé par l'interface et recherche si les coordonnées du point envoyé par l'interface se trouvent dans une case.

(3) soit le nombre de jetons sélectionnés est plus grand que 0.

(4) Demande d'un pointeur sur le premier jeton sélectionné.

(5) soit le mode de manipulation du jeton sélectionné est "sélection" + le(s) joueur(s) propriétaire(s).

(6) contrôle entre le joueur ayant la main et liste des joueurs propriétaires.

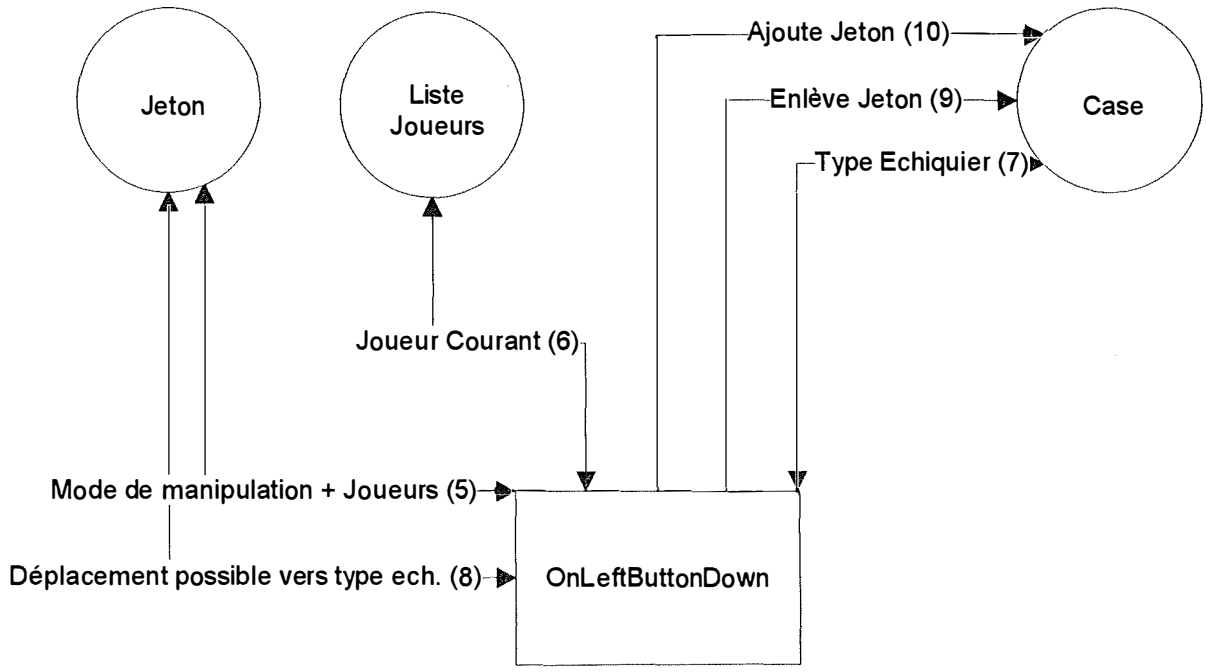
(7) Demande à la case sélectionnée, un pointeur sur son jeton visible.

(8) Contrôle du mode de manipulation du jeton visible. Puisque le (premier) jeton sélectionné est de type "sélection", le joueur clique dans une case pour y effectuer une autre sélection. Si le mode de manipulation du jeton visible n'est pas "sélection" il ne se passe rien (sortie de la fonction OnLeftButtonDown).

(9) Si l'objet visible est du type sélection, le message sélection est envoyé au jeton visible.

(10) Le message sélection est envoyé à l'objet règle du jeu. L'objet règle du jeu contrôle si le nombre de sélections à effectuer est atteint, si oui, il contrôle la validité de la sélection.

b2) Le mode de manipulation du jeton sélectionné est du type "déplacement" et le propriétaire de la case destination n'est pas un échiquier de type "grille"



(1) (2) (3) et (4) Ces messages sont les mêmes que au point (b1). Le message Un pointeur sur le premier jeton sélectionné est envoyé par l'objet règles du jeu.

(5) La fonction OnLeftButtonDown demande au jeton sélectionné son mode de manipulation, soit ici "déplacement" et une liste de pointeurs sur le(s) joueur(s) propriétaire(s).

(6) Si il y a au moins un propriétaire du jeton, il faut que le joueur ayant la main soit le propriétaire du jeton.

(7) La fonction OnLeftButtonDown demande à la case cliquée par le joueur (appelée case destination) et à la case dans laquelle se trouve le jeton sélectionné (appelée case origine), quel est le type d'échiquier de l'échiquier propriétaire de la case. Les types d'échiquiers possibles

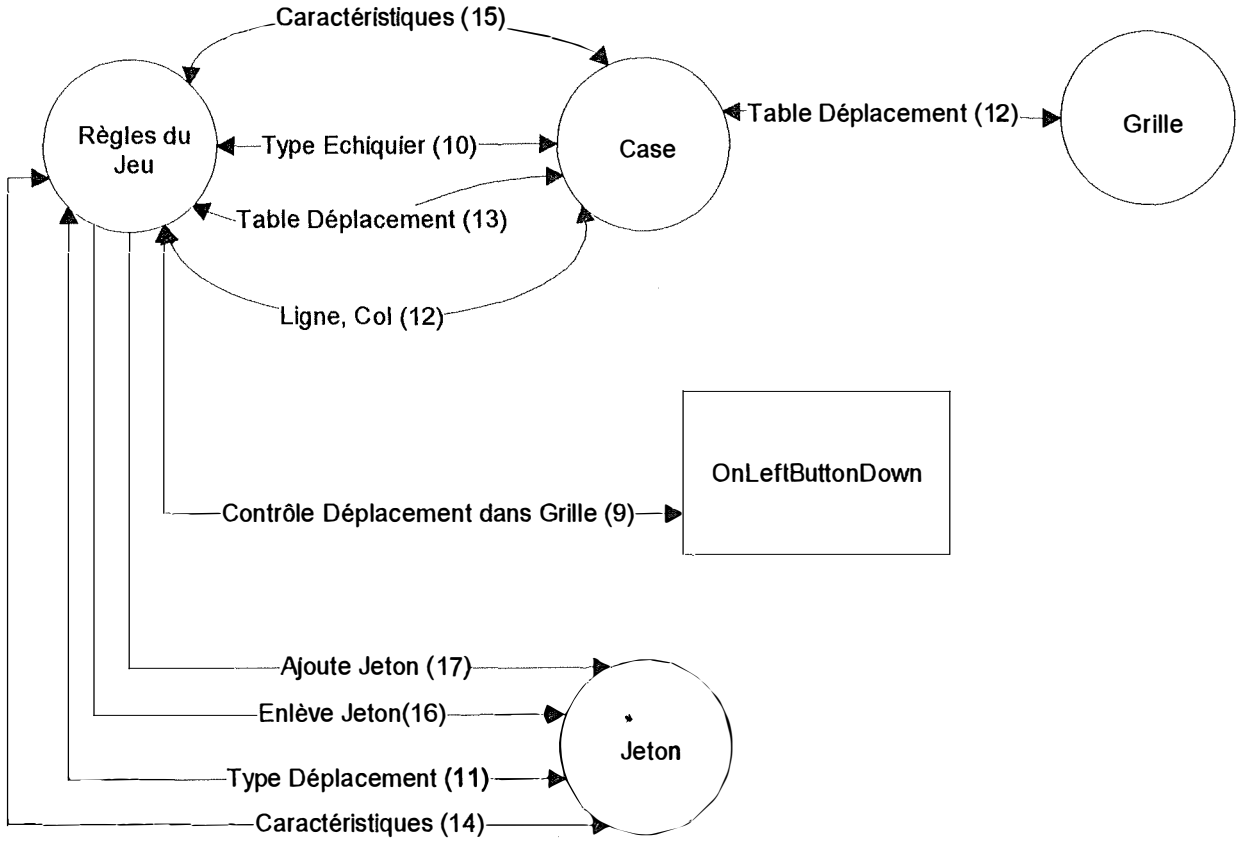
sont : la grille, le sabot, la zone de résultat et la zone d'attente. Nous supposons ici que le propriétaire de la case de destination ne soit pas de type "grille".

(8) La fonction OnLeftButtonDown demande au jeton si un déplacement entre les deux types d'échiquiers est possible. Si non, le jeton est désélectionné et la fonction se termine.

(9) Si il s'agit d'un déplacement vers la grille, il y a contrôle du déplacement par les règles du jeu (voir point b3). Ici le déplacement est effectué en enlevant le jeton de la case d'origine.

(10) En mettant le jeton dans la case de destination.

b3) Le mode de manipulation du jeton sélectionné est du type "déplacement" et le propriétaire de la case destination n'est pas un échiquier de type "grille"



Les messages (1), (2), (3), (4), (5), (6) (7) et (8) sont les mêmes que au point (b2). Le jeton sélectionné peut effectuer un déplacement du type échiquier de la case origine vers le type d'échiquier de la case de destination. Nous supposons ici que le type d'échiquier de la case de destination est "grille" message (7).

(9) La fonction OnLefButtonDown demande à l'objet règles du jeu de contrôler le déplacement à l'intérieur de la grille. Les paramètres fournis sont : un pointeur sur la case d'origine, un pointeur sur la case de destination et un pointeur sur le jeton.

(10) Contrôle si le type d'échiquier de la case d'origine est de type "grille"

si type = grille

(11) L'objet règle du jeu demande au jeton, quels sont ses déplacements à l'intérieur de la grille autorisés (Rappel : "tout permis", "une case", "x cases", "immobile").

(12) Pour les déplacements de "une case" et "x cases", l'objet règles du jeu a besoin de calculer la distance séparant la case d'origine et la case de destination. Pour calculer cette distance, il demande aux cases quelles sont leurs numéros de ligne et de colonne. Si la distance n'est pas valide ou si le jeton est immobile, l'objet règle du jeu retourne comme valeur à la fonction OnLefButtonDown : "deplacement_non_valide".

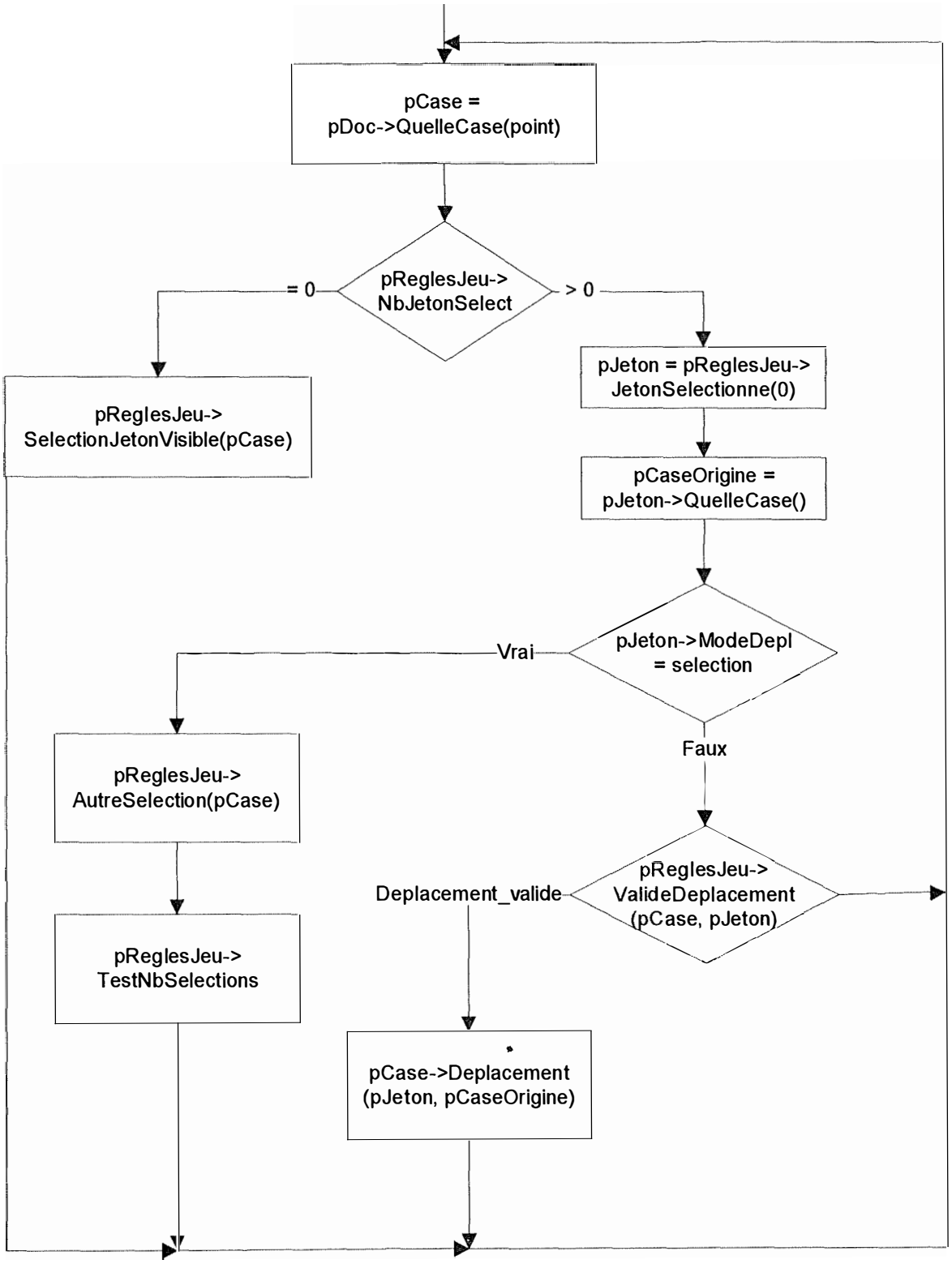
(13) Si le type de manipulation dans la grille autorisé est de "une case", l'objet règle du jeu demande à la case sa table de déplacement pour contrôler si la direction du déplacement est valide. Si il n'y a pas de table de déplacement associé à la case, la case renvoie la table de déplacement définie pour la grille. S'il n'y a pas de validation, valeur de retour = "deplacement_non_valide".

(14) L'objet règles du jeu demande au jetons ses caractéristiques. Les caractéristiques fournies sont des entiers (pointeurs dans la table des caractéristiques). Si au moins une caractéristique est fournie, il y a contrôle de caractéristiques avec celles de la case de destination.

(15) L'objet règle du jeu demande à la case de destination les caractéristiques de la zone dans laquelle il fait partie. La case pour répondre, répercute cette demande à la zone dont elle fait partie.

(16) et (17) S'il y a validation des caractéristiques du jeton et de la zone, l'objet règle du jeu génère un message Enlève Jeton à la case de la grille. A noter, que Enlève Jeton d'une case de grille est une fonction surchargée. En plus du traitement effectué par CCase::EnleveJeton, CCaseGr::EnleveJeton envoie un message EnleveCaractJeton à la zone de la case d'origine. Lorsqu'une zone reçoit ce message, elle enlève de ses caractéristiques, celles qui sont partagées uniquement par le jeton qui quitte la zone. L'objet règles du jeu génère également le message Ajouter Jeton destiné à la case de destination. De même, CCaseGr::AjouterJeton génère un message AjouteCaractJeton à la zone dont fait partie la case de destination.

En cas de validation, l'objet règle du jeu répond au message "Contrôle déplacement dans grille" par "deplacement_valide", en cas de non validation, l'objet règles du jeu répond le message "deplacement_non_valide".



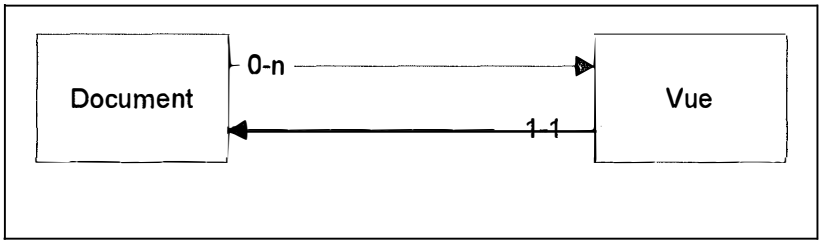
Messages générés lorsqu'un joueur clique dans une case

9. L'implémentation

9.1. Présentation

Comme je l'ai déjà mentionné, j'ai choisi d'utiliser le Visual C++ pour programmer le générateur de jeux. Il faut mentionner que le Visual C++ utilise une architecture particulière : l'architecture Document-Vue.

La classe Document stocke les données et gère la sérialisation, la classe Vue gère l'affichage et l'impression.. Cette architecture sépare les données de la vue qu'a l'utilisateur de ces données. Un avantage est de permettre à plusieurs vues différentes d'afficher les mêmes



données.

Les classes définies dans la partie conception doivent pouvoir sauver leurs données sur disque. Toutes ces classes vont avoir comme classe de base la classe Document. Chaque fois qu'une classe voudra afficher ses données, elle devra utiliser une classe dérivée de la classe Vue.

Chaque échiquier possède deux vues :

- une vue propre de l'échiquier, utilisée pour modifier les caractéristiques de l'échiquier. Cette vue est appelée "vue concepteur" car seul le concepteur d'un jeu à accès à cette vue.

- une vue du jeu, représentant la place de l'échiquier dans l'aire de jeu. Cette vue est appelée "vue joueur" car c'est la seule vue par le joueur.

Pour le générateur de jeux, le concepteur ne peut créer qu'un seul jeu à la fois. Il n'y a donc qu'une seule instance de la classe Document. Dans la conception du jeu, j'ai tiré parti de cette particularité. Tout objet obligatoire est imbriqué dans la classe Document. Dès lors, lors de la création d'un nouveau jeu, tout les objets obligatoires sont également créés avec leurs options par défaut.

Les objets obligatoires sont :

- caractéristiques
- liste sabots
- liste zones d'attente
- liste zones de résultat
- liste joueurs. Lors de la création de la liste des joueurs, il y a création automatique d'un joueur de nom "joueur".
- règles du jeu
- la grille concepteur, c'est la grille telle que définie par le concepteur (les jetons sont à la place choisie par le concepteur). Elle est utilisée pour créer la grille joueur lors de chaque nouvelle partie. Lors d'e la création d'une grille, il y a création automatique d'une zone "Reste Grille" contenant toutes les cases de la grille.
- la grille joueur, c'est la grille telle que vue par le joueur. Les jetons peuvent s'y trouver à une autre place, il peut y avoir plus ou moins de jetons. Création de la zone "Reste Grille".

Tous les autres objets sont hérités soit directement de Document, p. ex. la classe Echiquier ou indirectement, par ex. la classe Sabot (dérivée de Echiquier qui est à son tour dérivée de Document).

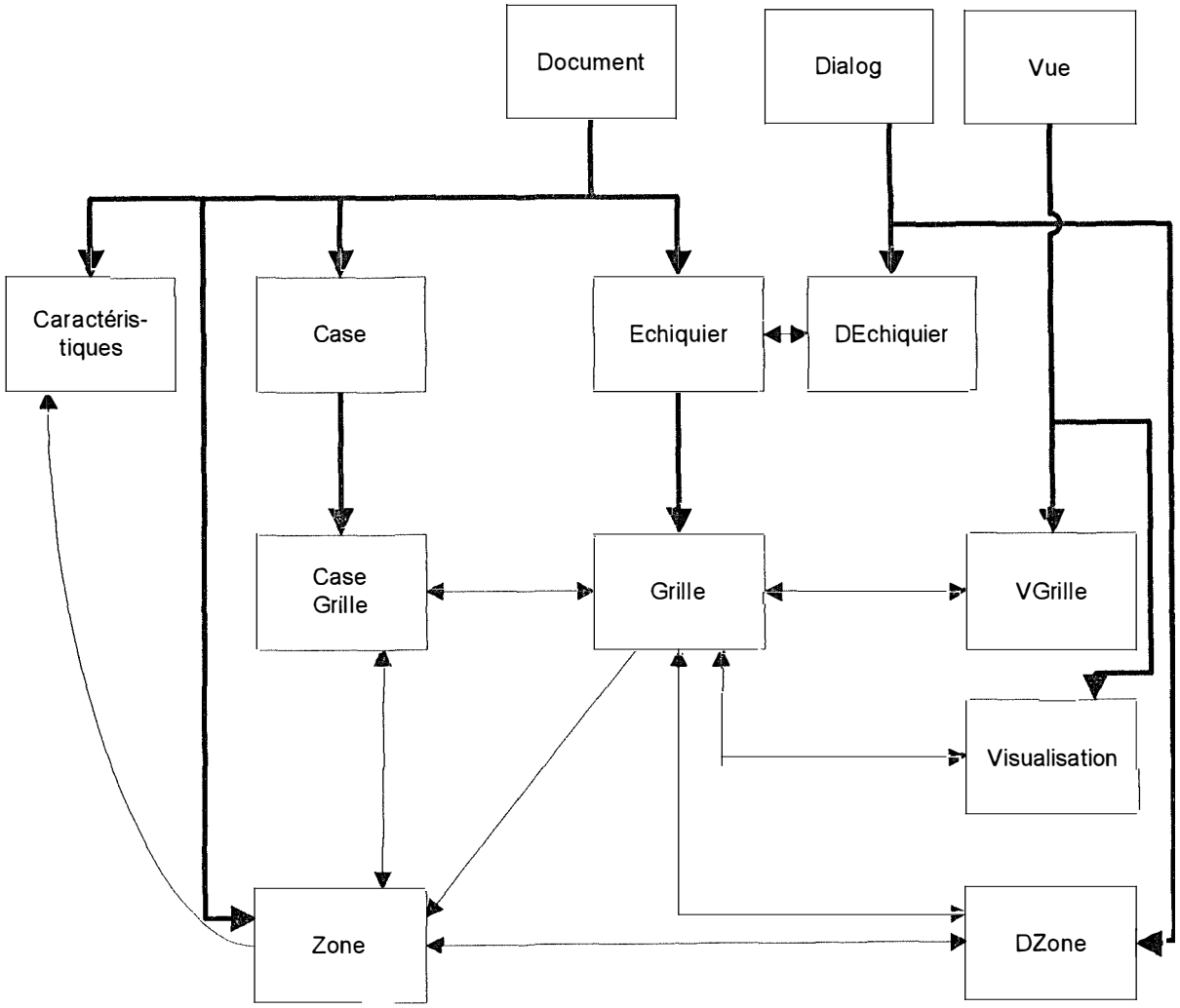
Pour chacune de ces classes, il va falloir définir une manière d'afficher les données. Il existe deux méthodes : afficher les données en permanence en créant une vue ou une boîte de dialogue non modale, afficher les données à un moment donné et utiliser une boîte de dialogue modale.

Il va falloir, revoir la découpe en objets effectuée, pour l'adapter à l'architecture Document-Vue. Chaque classe définie dans la partie conception va être une classe de l'objet document.

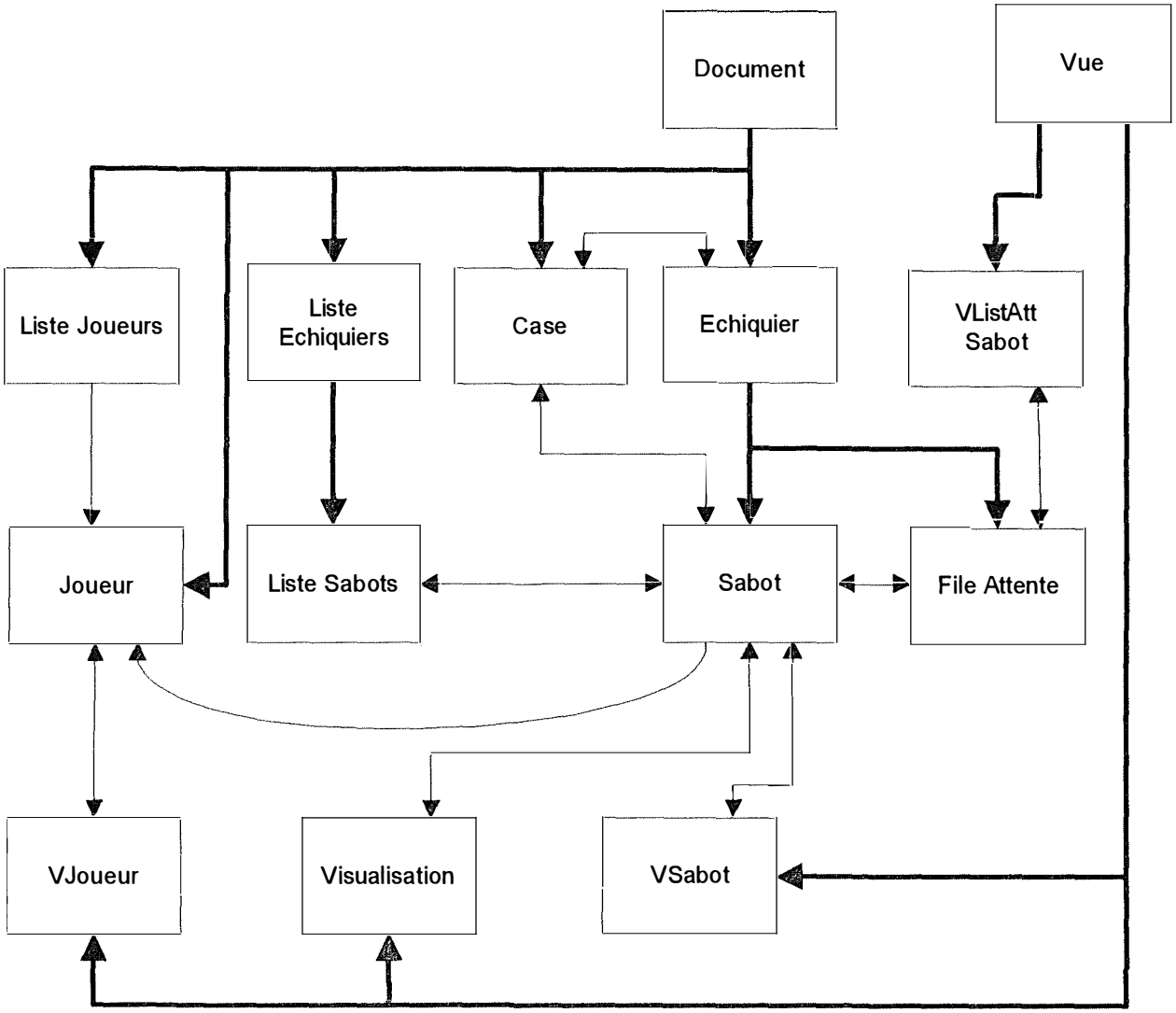
9.2. La grille

Comme nous l'avons vu, la grille est dérivée de la classe échiquier, et la classe échiquier est dérivée de la classe Document. La grille affiche ses données grâce à la classe VGrille dérivée de Vue. Montrons les relations existantes entre la classe grille et les autres classes.

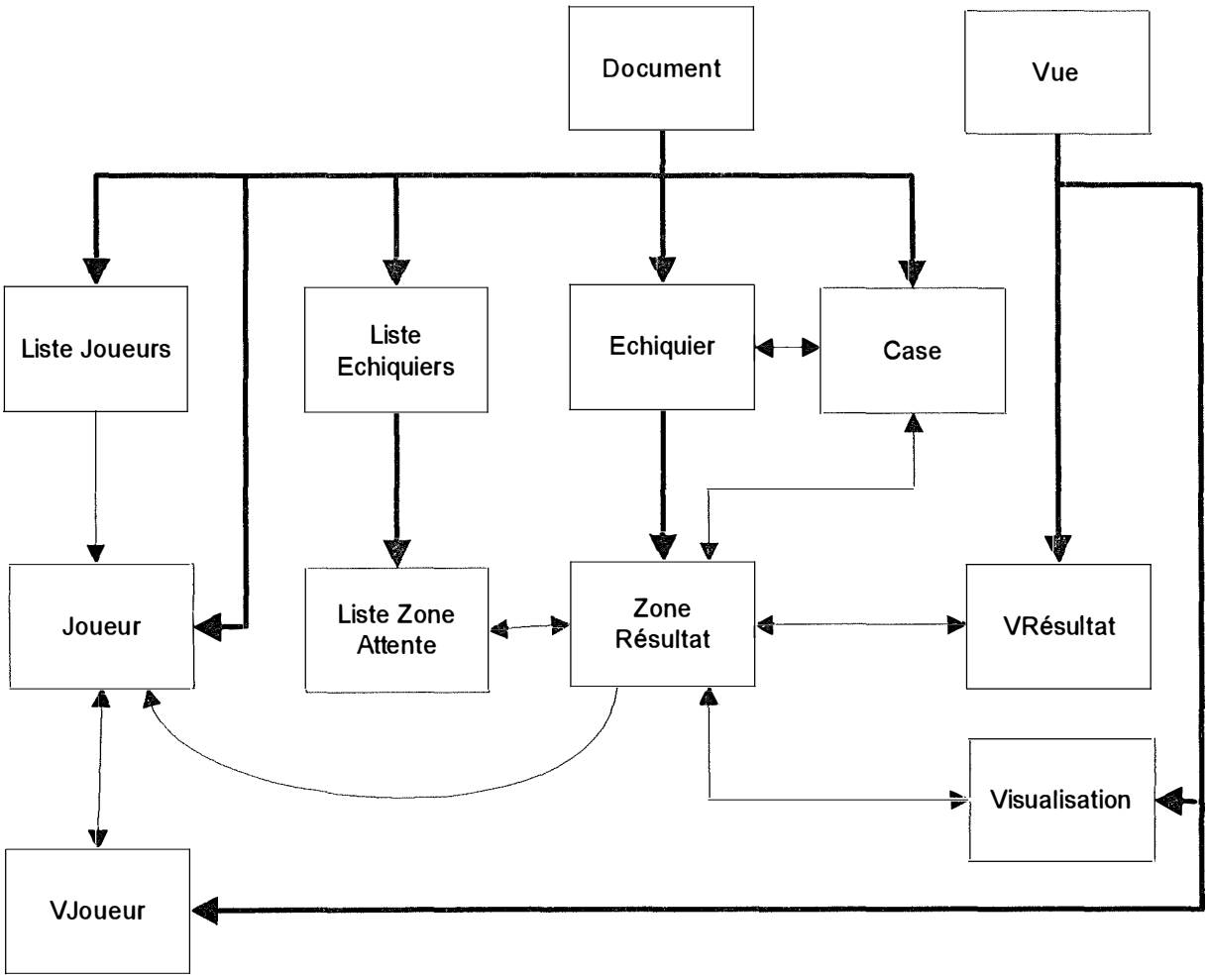
Notation : un trait épais indique une relation de filiation, un trait fin d'un objet A vers un objet B indique que A possède un pointeur vers B. Remarque : tout objet dérivé de la classe Vue possède un pointeur sur l'unique instance de la classe Document.


Document
Dialog
Vue
Caractéris-
tiques
Case
Echiquier
DEchiquier
Case
Grille
Grille
VGrille
Visualisation
Zone
DZone

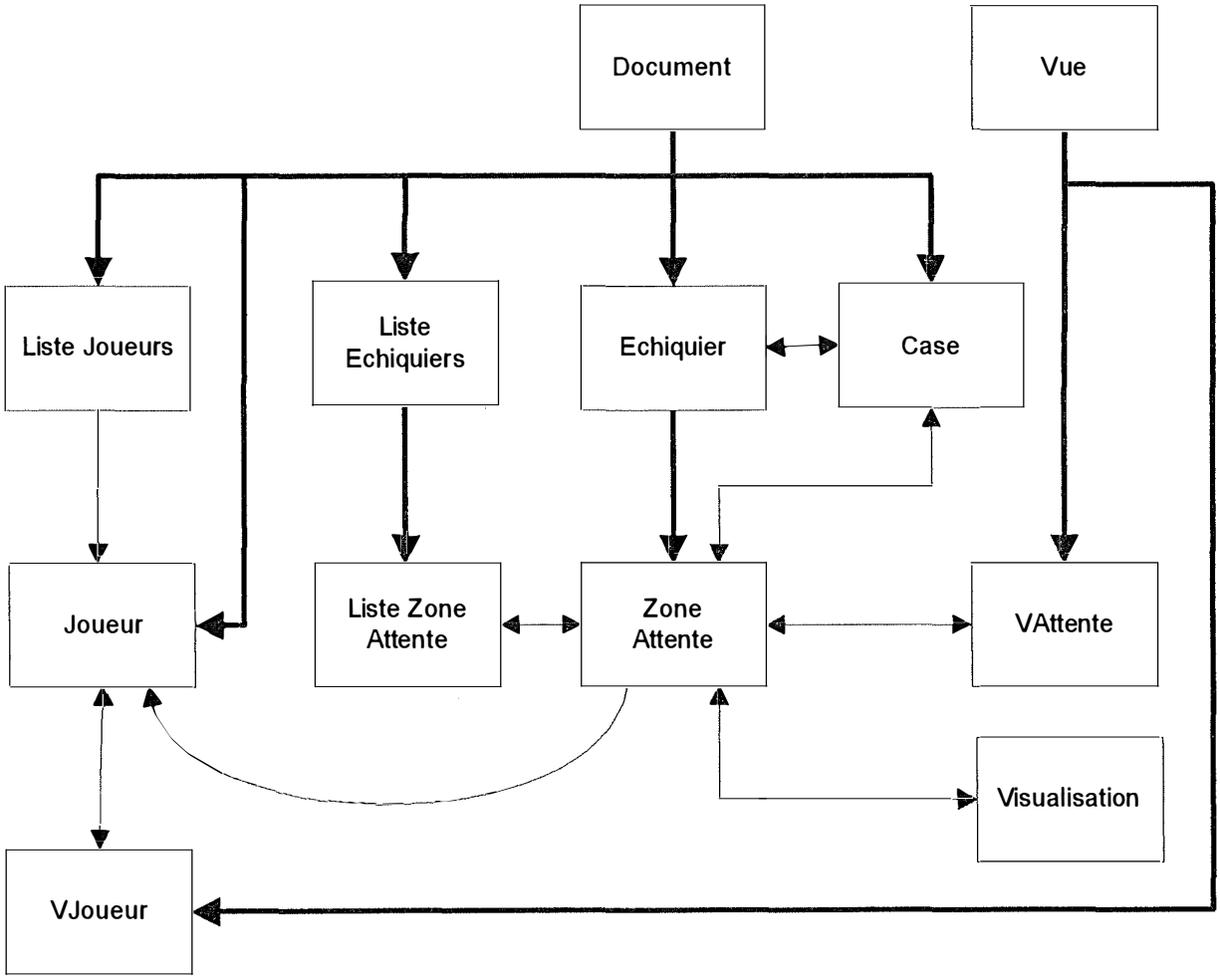
9.3. Le sabot


Document
Vue
Liste Joueurs
Liste Echiquiers
Case
Echiquier
VListAtt Sabot
Joueur
Liste Sabots
Sabot
File Attente
VJoueur
Visualisation
VSabot

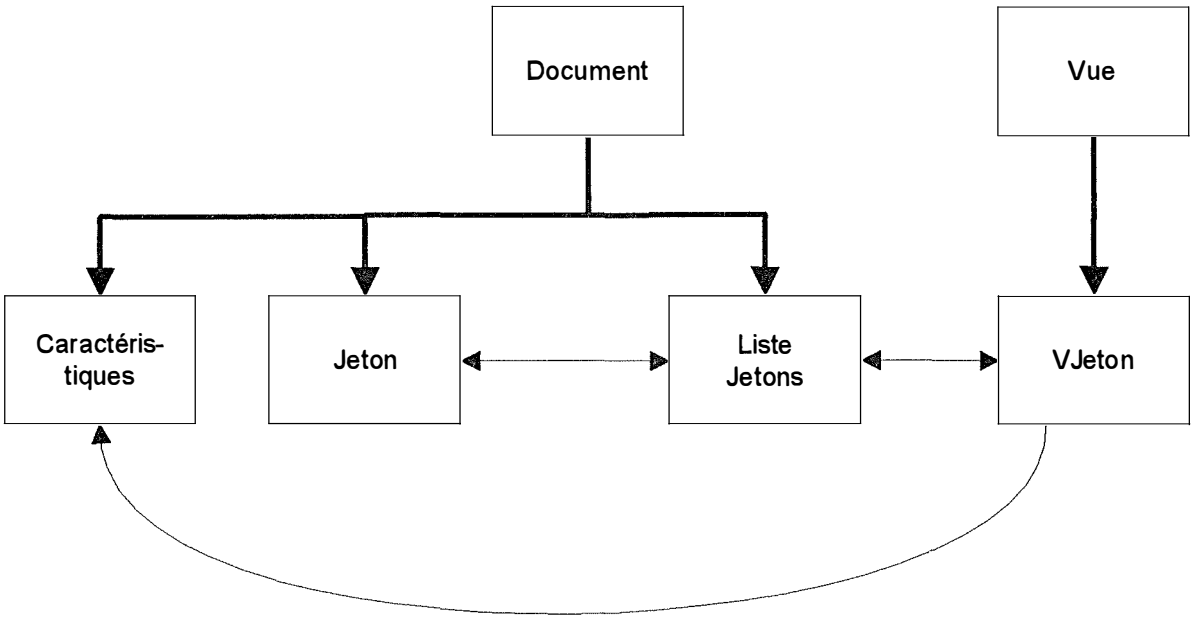
9.4. La zone de Résultat



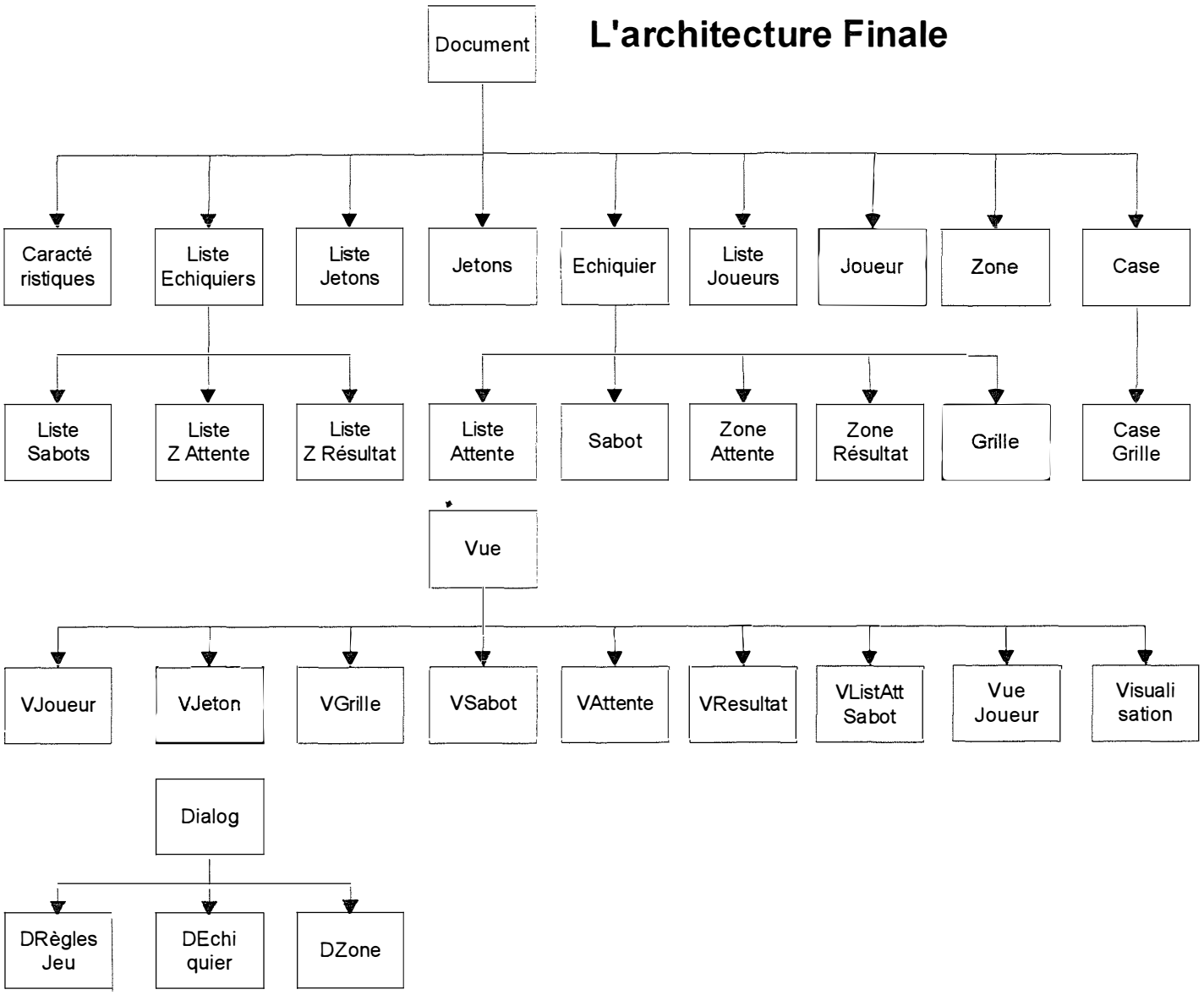
9.5. La zone d'Attente



9.6. Le jeton



Commentaire : Il n'y a pas de pointeur défini entre la classe Liste Jetons et la Classe Caractéristiques. En effet, la liste de caractéristiques d'un objet liste jetons est traitée comme un tableau d'entiers. Lors de la comparaison de caractéristiques, il y a comparaison d'entiers. La classe Vjeton doit posséder un pointeur sur les caractéristiques car c'est elle qui crée les caractéristiques et qui les affiche (le concepteur voit une caractéristique comme une chaîne de caractères). Chaque fois que l'on affiche une caractéristique à l'écran, il faut accéder à l'objet caractéristique gérant la table des caractéristiques).


L'architecture Finale
Document
Caracté ristiques
Liste Echiquiers
Liste Jetons
Jetons
Echiquier
Liste Joueurs
Joueur
Zone
Case
Liste Sabots
Liste Z Attente
Liste Z Résultat
Liste Attente
Sabot
Zone Attente
Zone Résultat
Grille
Case Grille
Vue
VJoueur
VJeton
VGrille
VSabot
VAttente
VResultat
VListAtt Sabot
Vue Joueur
Visuali sation
Dialog
DRègles Jeu
DEchi quier
DZone

Remarque : Chaque objet : grille, sabot, ... affiche ses données dans une vue : VGrille, vsabot. La vue "Vue Joueur" affiche les données telles qu'elles seront vues pendant la phase de jeu. La grille est immobile dans la vue VGrille mais peut être déplacée dans la vue Vue Joueur.

La classe règles du jeu n'affiche pas ses données dans une vue mais dans une boîte de dialogue. Ce choix a été fait de manière à pouvoir modifier les données de la règle du jeu à partir de n'importe quelle vue, sans devoir en changer.

10. Les fonctions offertes par les classes

Pour des raisons de clarté, je ne vais pas parler des classes définies dans la hiérarchie du Visual C++ mais bien de classes abstraites. Par exemple au lieu de CObArray, je parlerai de Liste Pointeur.

10.1. La classe Règle du jeu

Les constantes utilisées

VALIDATION_DEPLACEMENT VD : deplacement_valide, deplacement_non_valide, selection- jeton, enleve_jeton

DNV : deplacement_accepte, deplacement_refuse

DV accepter_deplacement, enlever_jeton

Les structures de données

Variable membre	Signification
BOOL m_caract_max	Validation des caractéristiques dans la grille par caractéristique max. ou min. ? Par défaut, validation par caractéristique maximum.
DNV m_dnv	Action à effectuer lors d'un déplacement (sélection) non validé (par caractéristique), les actions sont décrites par la constante DNV. Par défaut, refuser le déplacement.
string m_dnv_message	Message à afficher lorsqu'il n'y a pas validation par caractéristiques d'un déplacement dans la grille. Le message par défaut est vide.
DV m_dv	Action à effectuer lorsqu'il y a validation des caractéristiques : lors d'une sélection ou lors d'un déplacement dans la grille. Ces actions sont décrites par la constante DV. L'action par défaut est d'accepter le déplacement.
string m_dv_message	Message à afficher lorsque les caractéristiques d'un déplacement dans la grille ou d'une sélection sont validées. Le message par défaut est vide.
integer m_fin_x_depl	nombre maximum de déplacements, lorsque ce nombre est atteint le jeu est terminé. Par défaut ce nombre est 0.

integer m_fin_x_err	nombre maximum d'erreurs, lorsque ce nombre est atteint le jeu est terminé. Par défaut ce nombre est 0.
integer m_nb_selections	nombre de sélections à effectuer avant de contrôler les caractéristiques des jetons sélectionnés. Par défaut, 2.
BOOL m_fin_grille_remplie	indique si le jeu se termine si la grille est remplie
BOOL m_fin_grille_vide	indique si le jeu se termine si la grille est vide
BOOL m_fin_sabot_vide	indique si le jeu se termine si le sabot est vide
BOOL m_fin_zones_valides	indique si le jeu se termine lorsqu'il y a validation de tous les jetons de la grille avec les caractéristiques "concepteur" des zones où ils se trouvent
BOOL m_fin_zones_valides_sabot_vide	indique si le jeu se termine lorsque tous les jetons de la grille sont validés et lorsque le sabot est vide
BOOL m_fin_zones_valides_grille_remplie	indique si le jeu se termine lorsque tous les jetons de la grille sont validés et lorsque la grille est remplie
BOOL m_permettre_deselection	lorsque le mode de manipulation d'un jeton est sélection et qu'il est sélectionné, peut-il être désélectionné ou non ?
BOOL m_finjoueur_finjeu	indique si la fin d'un joueur entraîne la fin du jeu
BOOL m_b_gagne	indique s'il faut afficher un message lorsque le jeu est gagné (les conditions de fin de jeu sont remplies)
string m_message_gagne	message à afficher lorsque le jeu est gagné
BOOL m_b_perdu	indique s'il faut afficher un message lorsque le jeu est perdu
string m_message_perdu	message à afficher lorsque le jeu est perdu
BOOL m_tour_boite_dialogue	indique s'il faut afficher une boîte de dialogue tour de jeu, permettant à un joueur de passer la main au joueur suivant
BOOL m_tour_deplacement	indique si la main change après un déplacement
BOOL m_tour_deplacement_valide	indique si la main change après un déplacement valide
BOOL m_attendre_x_secondes	indique s'il faut attendre lorsqu'une sélection n'est pas validée avant de retourner les jetons sélectionnés
integer m_x_secondes	nombre de 1/10 de secondes à attendre lorsqu'une sélection n'est pas validée
BOOL m_message_select_non_valide	indique s'il faut afficher une boîte de dialogue lorsqu'une sélection n'est pas validée, avant de retourner les jetons sélectionnés
string m_selection_non_valide	message à afficher dans la boîte de dialogue sélection non valide
BOOL m_message_select_valide	indique s'il faut afficher une boîte de dialogue lorsqu'une sélection est validée, avant d'enlever les jetons

string m_selection_valide	message à afficher dans la boîte de dialogue sélection valide
CObArray m_jetons_selectionne	liste de pointeurs sur des jetons sélectionnés

Les services offerts

Fonction	Objectif
void AjouteDeplacement()	Incrémente le nombre de mouvements du joueur qui a la main
void AjouteErreur()	Incrémente le nombre d'erreurs du joueur qui a la main
BOOL ControleFinJeu()	Teste si les conditions de fin de jeu sont remplies, fait appel à JeuGagne et JeuPerdu
BOOL JeuGagne()	Contrôle si le jeu est gagné.
BOOL JeuPerdu()	Contrôle si le jeu est perdu.
TourJeuSuivant(VD)	Reçoit comme paramètre le résultat du contrôle effectué sur le déplacement (valide, non valide, sélection). La procédure teste s'il faut envoyer le message JoueurJeuSuivant à l'objet ListeJoueurs. TourJeuSuivant affiche ensuite dans le nom du joueur courant dans la zone de messages.
VD TestNbSelections()	Si le nombre de jetons à sélectionner est atteint, cette fonction demande le contrôle des caractéristiques, affiche éventuellement les boîtes de dialogue (sélection validée ou sélection non validée) et renvoie un message de type VD.
void EnleveJetonsSelectionnes()	Pour tous les jetons sélectionnés, enlève les jetons de l'échiquier courant et les pose dans la zone de résultat du joueur courant (si disponible). La liste des jetons sélectionnés est ensuite vidée.
void SelectionJetonVisible(case*)	Le jeton visible de la case pointée est ajouté à la liste des jetons sélectionnés et il reçoit le message Selection()
void AutreSelectionJeton(case*)	Cette fonction est appelée lorsque au moins un jeton est sélectionné. Elle désélectionne (éventuellement) un jeton déjà sélectionné, ou sinon envoie le message SelectionJetonVisible à la case référencée.
void DeSelectionJetons()	Envoie à tous les jetons de la liste des jetons sélectionnés le message Désélection et ensuite vide la liste des jetons sélectionnés.
void DeSelectionneJeton(int i)	Enlève le jeton numéro i de la liste des jetons sélectionnés.
CJeton* JetonSelectionne(int i)	Retourne le jeton sélectionné numéro i.
BOOL ValidationJetons()	Contrôle la validation des jetons sélectionnés. Pour ce faire, elle utilise la fonction "validation".
BOOL Validation(Liste Entier*,	Contrôle de la validation (par caract. max. ou min.) entre

Liste Entier*)	les caractéristiques des deux listes fournies.
VD ValideDeplacement(case*, case*, jeton*)	Contrôle si le déplacement effectué par le jeton pointé entre la case origine et la case de destination est valide. Le résultat est du type VD (VALIDATION_DEPLAC).
void AttendreXSecondes(int x)	Fonction qui attend x 1/10 de secondes.
void Deplacement(VD)	Envoie des messages : AjouteDeplacement, éventuellement AjouteErreur et TourJeuSuivant.
BOOL DeplacementEntreEchiquiersValide(case*, case*, jeton*)	Contrôle si le jeton pointé peut se déplacer entre l' échiquier propriétaire de la case origine et l'échiquier propriétaire de la case de destination.
BOOL ValideDistanceDirectionGrille(case *, case*, jeton*)	Lors d'un déplacement à l'intérieur de la grille. Fait (éventuellement) appel à CalculDistance et DirectionValide pour valider un déplacement à l'intérieur de la grille.
int CalculDistance(case*, case*)	Calcul de la distance séparant deux cases d'un même échiquier.
BOOL DirectionValide(caseGr*, caseGr*)	Teste si la direction d'un déplacement d' une case est valide en fonction de la table de déplacement de la case origine de la grille (ou éventuellement de la table de déplacement de la grille).
BOOL ValideDepGrille(case*, jeton*)	Teste s'il y a validation entre les caractéristiques d'un jeton et les caractéristiques de la zone de la case de destination. Ce test est fait par un appel à "Validation"

10.2. La classe Echiquier

Les constantes utilisées

VUE : vue_concepteur, vue_joueur

TYPE_ECHIQUIER : grille, sabot, attente, resultat

Les structures de données

Variable membre	Signification
TYPE_ECHIQUIER m_type_echiquier	type de l'échiquier
string m_nom	nom de l'échiquier, ce nom est utilisé par exemple lorsque l'on demande au concepteur de choisir un sabot d'un joueur parmi plusieurs. Pour chaque sabot, on fournit le nom du sabot, par défaut type de l'échiquier.
point m_debut	coordonnées du coin supérieur gauche de

	l'échiquier, par défaut 0,0.
int m_nb_lignes	nombre de lignes, par défaut 0.
int m_nb_col	nombre de colonnes, par défaut 0.
int m_h_lignes	hauteur des lignes, par défaut 50.
int m_l_col	largeur des colonnes, par défaut 50.
int m_e_lignes	espace entre les lignes, par défaut 0.
int m_e_col	espace entre les colonnes, par défaut 0.
int m_epais_trait	épaisseur du trait, par défaut 1.
int m_nb_jetons	nombre de jetons se trouvant dans l'échiquier, par défaut 0.
string m_cheminDIB	chemin du fichier contenant le dessin de l'échiquier.
Liste Pointeur m_tab_cases	liste de pointeurs sur les cases de l'échiquier, par défaut liste vide.
Liste Pointeur m_proprietaires	liste de pointeurs sur les joueurs propriétaires de l'échiquier.

Les services offerts

Fonction	Objectif
BOOL EchiquierRemplit()	Contrôle si chaque case de l'échiquier possède au moins un jeton.
BOOL EchiquierVide()	Retourne vrai si le nombre de jetons = 0 sinon retourne faux.
BOOL ContientPoint(point)	Contrôle si une case de l'échiquier possède les coordonnées du point.
AjouteLigne(int ligne)	Ajoute une ligne de numéro ligne. Si cette ligne existe déjà, il convient de renuméroter les lignes existantes avant d'appeler cette fonction.
AjouteCol(int col)	Ajoute une colonne. idem AjouteLigne.
BOOL SupprimeLigne(int i)	Supprime la ligne de numéro i, si une case de cette ligne possède un jeton, la ligne n'est pas supprimée.
BOOL SupprimeCol(int i)	Supprime la colonne de numéro i. Si une case de la colonne possède un jeton, la colonne n'est pas supprimée.
RenumeroterPlusLignes(int ligne)	Incrémente les numéros de ligne de toutes les lignes d'un échiquier à partir du numéro de ligne donné.
void RenumeroterPlusCol(int col)	Incrémente les numéros de colonne d'un échiquier à partir du numéro donné.
void RenumeroterMoinsLignes(int ligne)	Décrémente les numéros de ligne ...
void RenumeroterMoinsCol(int col)	Décrémente les numéros de col ...

ChangementTailleCases(int haut_cases, int larg_cases)	change la taille des cases.
ChangementEspEntreCases(int, int)	change l'espace entre deux cases.
void Dessine(view*, VUE)	dessine l'échiquier dans une vue, le paramètre VUE indique s'il faut dessiner l'échiquier en (0-0) pour la vue concepteur ou en m_debut pour la vue joueur.
case* QuelleCaseEnPoint(point)	fournit un pointeur sur la case de l'échiquier contenant les coordonnées du point donné.
case* QuelleCase(int ligne, int col)	fournit un pointeur sur la case la ligne ligne et de la colonne col de l'échiquier.
void VideEchiquier()	Enlève tous les jetons se trouvant dans l'échiquier.
AjouteJeton()	Incrémente m_nb_jetons.
EnleveJeton()	Décrémente m_nb_jetons.
MelangerEchiquier(echiquier* , echiquier*)	copie les jetons de l'échiquier source vers l'échiquier de destination en mélangeant les jetons de l'échiquier source (sauf les jetons immobiles).
CopierEchiquier(Echiquier* , Echiquier*)	copie les jetons de l'échiquier source vers l'échiquier de destination en gardant les jetons aux mêmes endroits.
AjouteJoueur(Joueur*)	Ajoute un pointeur sur un joueur à la liste des joueurs propriétaires de l'échiquier.
NbJoueurs()	Retourne le nombre de joueurs propriétaires de l'échiquier (soit la taille de la liste des joueurs propriétaires).
SupprimeJoueur(Joueur*)	Enlève le joueur pointé de la liste des joueurs propriétaires de l'échiquier.

10.3. La classe grille

Les structures de données

Variable membre	Signification
BOOL m_ordre_aleatoire	Indique, lors de la copie de la grille concepteur vers la grille jeu, si les jetons doivent être mélangés ou non, par défaut les jetons sont mélangés.
BOOL m_hg	Il s'agit d'un élément de la table de déplacement de la grille. Il indique si un déplacement d'un jeton se trouvant dans une case de la grille peut effectuer un déplacement d' une case vers la case se

	trouvant en haut à gauche, par défaut non.
BOOL m_h	indique si le déplacement peut se faire vers le haut, par défaut oui.
BOOL m_hd	indique si le déplacement peut se faire vers le haut à droite, par défaut non.
BOOL m_g	indique si le déplacement peut se faire vers la gauche, par défaut oui.
BOOL m_c	indique si la case peut être une case de destination d'un déplacement, par défaut oui.
BOOL m_d	indique si le déplacement peut se faire vers la droite, par défaut oui.
BOOL m_bg	indique si le déplacement peut se faire vers le bas à gauche, par défaut non.
BOOL m_b	indique si le déplacement peut se faire vers le bas, par défaut oui.
BOOL m_bd	indique si le déplacement peut se faire vers le bas à droite, par défaut non.
Liste Pointeur m_zone;	liste de pointeurs sur des objets zone.
int m_pt_zone	pointe sur un pointeur de la liste m_zone, la zone m_zone(m_pt_zone) est la zone courante.

Les services offerts

Fonction	Objectif
void SupprimeContenu()	Enlève tous les jetons (via VideEchiquier), supprime toutes les cases et toutes les zones (via SupprimeToutesLesZones)
void AjouteJeton(case*, jeton*)	Appel à CEchiquier::AjouteJeton et envoie le message AjouteCaractJeton à la zone de la case pointée. Lorsqu'une zone reçoit ce message, elle ajoute les caractéristiques à sa liste de caractéristiques.
void EnleveJeton(case* , jeton*)	Appel à CEchiquier::EnleveJeton et envoi du message EnleveCaractJeton.
void SupprimeToutesLesZones()	Supprime toutes les zones de la grille.
void SupprimeZoneCourante()	Supprime la zone courante, pointée par m_pt_zone.
Zone* NouvelleZone()	création d'une nouvelle zone et ajout du pointeur vers cette zone dans le tableau des zones et retourne un pointeur vers cette zone.
Zone* ChangeZoneCourante(int nb)	Changement du pointeur de zone courante, une nouvelle zone est la zone courante.
Zone* Zone(int nb)	Entrée nb, sortie retourne la $nb^{ième}$ zone de la liste des zones.
Zone* ZoneCourante	Retourne un pointeur sur la zone courante.
Zone* QuelleZone(CCase*)	Retourne un pointeur sur la zone de la case

	pointée.
caseGr* QuelleCaseEnPoint(point point, point debut)	Entrée un point, Sortie un pointeur sur une case de la grille ou NULL.
caseGr* QuelleCase(int ligne, int col)	Entrée un numéro de ligne et un numéro de colonne, Sortie un pointeur sur une case de la grille ou NULL.
caseGr* Case(int i)	Entrée un entier, Sortie un pointeur sur la $i^{ème}$ case du tableau des cases de la grille.
BOOL SupprimeLigne(int ligne)	Fonction surchargée, car dans la grille lorsque l'on supprime une ligne, il faut enlever les cases supprimées des zones de la grille.
BOOL SupprimeCol(int col)	Idem lorsque l'on supprime une colonne.
AjouteLigne(int ligne)	Lorsque l'on ajoute une ligne, il faut ajouter toutes les cases de la ligne dans la zone "Reste de la grille".
AjouteCol(int col)	idem.
BOOL ToutesZonesValidees()	Teste si les jetons se trouvant dans les zones sont valides par rapport aux caractéristiques concepteur des zones.
static CopierGrille(grille*, grille*);	Destruction de la grille de destination (via SupprimeContenu), création d'une grille de même dimension, copie de la table de déplacement et des coordonnées, appel à CopierJetonsGrille.
static CopierJetonsGrille(grille*, grille*);	Recopie les jetons de la grille source vers la grille destination. Cette copie se fait soit en gardant l'ordre des jetons, soit en mélangeant les jetons qui ne sont pas immobiles.

10.4. La classe "sabot"

Les structures de données

Variable membre	Signification
ListeAtt* mp_liste_att	Pointeur sur la liste d'attente du sabot. Cette liste contient les jetons placés par le concepteur.
ListeAtt* mp_liste_att_jeu	Pointeur sur la liste d'attente du sabot utilisée pour le jeu. A chaque nouvelle partie, il y a copie des jetons de la liste d'attente concepteur vers la liste d'attente joueur (ou jeu). Cette copie se fait soit en gardant la place des jetons, soit en mélangeant les jetons.
Liste Pointeur m_proprietaires	Liste de pointeurs sur les joueurs propriétaires du sabot.

Les services offerts

Fonction	Objectif
DeplacementJetonSabot()	Lors d'un déplacement validé, si le propriétaire de la case origine du déplacement est le sabot, ce message est envoyé au sabot. Lorsque un sabot reçoit ce message, il regarde quel est son mode de remplissage et envoie ou non un message à sa liste d'attente pour demander de remplir le sabot ou demande l'exécution de la fonction RemplitUneCase().
RemplitUneCase()	Cette fonction parcourt le sabot à la recherche d'une case vide. Si une case vide est trouvée, elle demande à la liste d'attente le jeton suivant et l'ajoute à la case trouvée.

10.5. La classe liste d'attente du sabot

Les structures de données

Variable membre	Signification
BOOL m_ordre_aleatoire	Indique si la copie des jetons dans la liste d'attente jeu doit se faire en mélangeant les jetons ou non.
int m_pt_liste_att	Pointeur sur le prochain jeton à copier

Les services offerts

Fonction	Objectif
RemplirSabot()	Parcourt les cases du sabot, chaque fois que l'on trouve une case vide on copie le jeton suivant de la liste d'attente. On s'arrête dès qu'il n'y a plus de jetons dans la liste d'attente ou que toutes les cases du sabot sont remplies
CopierListeAtt(Liste Attente*)	Copie la liste pointée vers la liste courante, éventuellement en mélangeant les jetons.
sabot* mp_sabot	Pointeur sur le sabot propriétaire de la liste d'attente.
jeton* JetonSuivant()	Parcourt la liste d'attente, à partir de la position pointée par m_pt_liste_att, à la recherche d'une case non vide. Lorsque une case possédant a été trouvée, mise à jour de m_pt_liste_att et retourne un pointeur sur le jeton

	trouvé dans la case.

10.6. La classe zone d'attente

Les structures de données

La zone d'attente n'a pas d'autres données que celles héritées de la classe échiquier.

Les services offerts

Ceux offerts par la classe échiquier.

10.7. La classe zone de résultat

Les constantes utilisées

stylerempzresultat : decaler_ligne, decaler_colonne, vide, rien

Les structures de données

Variable membre	Signification
stylerempzresultat m_style_remp	Style de remplissage de la zone de résultat lorsque la zone est remplie et qu'il faut afficher un nouveau jeton.
BOOL m_remplissage_ligne	Le remplissage de la zone se fait ligne par ligne ou colonne par colonne ?

Les services offerts

Fonction	Objectif
AjouteJeton(Jeton*)	Demande à la zone de résultat d'afficher un nouveau jeton.
ZoneResultatRemplie()	Teste si la zone est remplie.
DecaleLignes()	Supprime les jetons se trouvant dans la première ligne et affiche tous les jetons sur la ligne précédente.

DecaleCol()	Supprime les jetons se trouvant dans la première colonne et affiche tous les jetons sur la colonne précédente.

10.8. La classe liste d'échiquiers

Les structures de données

Variable membre	Signification
Liste Pointeur m_liste_echiquier	Liste de pointeur sur des échiquiers
int m_pt_echiquier	Pointe sur un échiquier de la liste de pointeurs m_liste_echiquier. Cet échiquier est l'échiquier courant de la liste d'échiquiers.

Les services offerts

Fonction	Objectif
AjouteEchiquier(Echiquier*)	Ajoute un pointeur sur un échiquier à la liste de pointeurs.
SupprimeEchiquier(Echiquier*)	Supprime un pointeur de la liste de pointeurs.
int QuelPtListe(Echiquier*)	Retourne l'indice dans la liste de pointeurs duquel se trouve l'échiquier donné en paramètre.
VideListeEchiquier()	Pour tous les échiquiers de la liste, envoie le message VideEchiquier. Enlève les jetons des échiquiers de la liste .
DeleteContents()	Appel de la fonction DeleteContents de chaque échiquier.
Dessine(Vue*)	Dessin d'un échiquier.
Echiquier* Echiquier(int)	Retourne un pointeur sur le $i^{ème}$ échiquier de la liste.
Echiquier* ContientPoint(Point)	Retourne un pointeur sur l'échquier de la liste dont une des cases contient les coordonnées du point donné en paramètre. Si aucune case des échiquiers de la liste ne contient ces coordonnées, la fonction retourne NULL.
Echiquier* ChoixEchiquierNom()	Affiche une boîte de dialogue contenant les noms de la liste. Si le joueur choisit un nom d'échiquier, la fonction retourne un pointeur vers cet échiquier, sinon elle retourne NULL.

10.9. La classe liste de sabots

Les constantes utilisées

STYLE_REMPLISSAGE : quitte_sabot, sabot_vide, demande

Les structures de données

Variable membre	Signification
STYLE_REMPLISSAGE m_style_remplissage	Options de remplissage des sabots de la liste de sabots.

Les services offerts

Fonction	Objectif
sabot* NouveauSabot()	Crée un nouveau sabot, met à jour la liste des sabots et retourne un pointeur sur le sabot créé.
sabot* Sabot(int i)	Retourne le ième sabot de la liste.
sabot* QuelSabotID(int id_sabot)	Retourne le sabot de nom id_sabot
InitListAttSabot()	Enlève les jetons de tous les sabots et remet le pointeur de la liste d'attente du sabot à zéro.

10.10. La classe liste des zones de résultat

Les structures de données

La classe liste des zones de résultat n'a pas d'autres variables membres que celles héritées de la liste d'échiquiers.

Les services offerts

Fonction	Objectif

resultat* NouvelleZResultat()	Crée une nouvelle zone de résultat, met à jour la liste des zones de résultat et retourne un pointeur sur la zone de résultat créée.

10.11. La classe liste des zones d'attente

Les structures de données

La classe liste des zones de d'attente n'a pas d'autres variables membres que celles héritées de la liste d'échiquiers.

Les services offerts

Fonction	Objectif
attente* NouvelleZAttente()	Crée une nouvelle zone d'attente, met à jour la liste des zones d'attente et retourne un pointeur sur la zone de résultat créée.

10.12. La classe liste joueurs

Les structures de données

Variable membre	Signification
Liste Pointeur m_liste_joueurs	Liste de pointeurs vers les joueurs créés par le concepteur.
Liste Pointeur m_liste_joueurs_jeu	Liste de pointeurs vers les joueurs du jeu.
int m_pt_joueur	Pointeur sur le joueur "concepteur" courant.
int m_pt_joueur_jeu	Pointeur sur le joueur "jeu" courant.

Remarques : Il y a deux listes de pointeurs vers des joueurs. La "liste concepteur" contient les données définies par le concepteur. La "liste concepteur" pointe sur des joueurs statiques. Il s'agit des joueurs définis lors de la création du jeu et servant de base à la construction dynamique des joueurs. Le nom de chaque joueur pointé par cette liste est le nom défini par le concepteur, les jetons de la file d'attente du sabot sont rangés d'après l'ordre défini par le concepteur. La "liste joueur" pointe sur des joueurs créés dynamiquement. Les noms des joueurs sont créés interactivement, les files d'attente du sabot peuvent avoir été mélangées,

le nombre de joueurs de la liste joueur peut être plus grand que le nombre de joueur de la liste concepteur.

Deux pointeurs sont utilisés : un pour la liste "concepteur" et un pour la liste "jeu". Rappelons qu' à tout moment, le concepteur peut, lors du test d'un jeu, quitter le jeu qu'il teste, effectuer des modifications et continuer le jeu en cours.

Les services offerts

Fonction	Objectif
DeleteContents()	Supprime les joueurs pointés par les listes "concepteur" et "jeu".
BOOL SupprimeJoueurCourant()	Le concepteur ne peut supprimer un joueur que si ce joueur n'est propriétaire d'aucun échiquier (sabot, zone d'attente ou zone de résultat).
SupprimeJoueurJeuCourant()	Lors d'un jeu, on peut ajouter de nouveaux joueurs et/ou en supprimer. Ici, il n'y a pas de valeur de retour car c'est l'interface qui effectue le contrôle. Si le joueur "jeu" courant peut être supprimé, l'interface active le bouton supprimer joueur, s'il ne peut pas être supprimé, ce bouton est désactivé.
Joueur* QuelJoueurCourant()	Retourne un pointeur sur le joueur "concepteur" courant.
Joueur* QuelJoueurJeuCourant()	Retourne un pointeur sur le joueur "jeu" courant.
Joueur* JoueurJeuSuivant()	Retourne un pointeur sur le joueur "jeu" suivant.
Joueur* Suivant()	Retourne un pointeur sur le joueur "concepteur" suivant.
Joueur* Precedent()	Retourne un pointeur sur le joueur "concepteur" précédent.
Joueur* Premier()	Retourne un pointeur sur le premier joueur de la liste "concepteur".
Joueur* Dernier()	Retourne un pointeur sur le dernier joueur de la liste "concepteur".
Joueur* NouveauJoueur()	Crée un nouveau joueur "concepteur", met à jour la liste "concepteur" et renvoie un pointeur sur le joueur créé.
Joueur* NouveauJoueurJeu()	idem que pour la liste "jeu". Notons que la création d'un joueur "jeu" n'est autorisée que s'il existe un seul joueur "concepteur". Dans ce cas, le joueur "jeu" est initialisé avec les données du joueur "concepteur".
Joueur* Joueur(int i)	Retourne le $i^{ème}$ joueur de la liste "concepteur".
Joueur* JoueurJeu(int i)	Retourne le $i^{ème}$ joueur de la liste "jeu".
int NbJoueurs()	Retourne le nombre de joueurs de la liste "concepteur".
int NbJoueursJeu()	Retourne le nombre de joueurs de la liste "jeu".
CopierJoueursJoueursJeu()	Crée la liste "jeu" et les joueurs "jeu".

10.13. La classe joueur

Les structures de données

Variable membre	Signification
Attente* mp_attente_joueur	Pointeur sur la zone d'attente utilisée par le joueur.
Resultat* mp_resultat_joueur	Pointeur sur la zone de résultat utilisée par le joueur.
Sabot* mp_sabot_joueur	Pointeur sur le sabot utilisé par le joueur.
string m_nom_joueur	Nom du joueur.
int m_nb_erreurs	Nombre d'erreurs que le joueur a faites.
int m_nb_mouvements	Nombre de mouvements que le joueur a faites.
BOOL m_joue	Indique si le joueur joue encore ou s'il a été mis en sommeil.

Remarque : Les variables membres m_nb_erreurs, m_nb_mouvements et m_joue ne sont utilisées que pour des joueurs "jeu".

Les services offerts

Fonction	Objectif
AugmenteErreurs()	Incrémente m_nb_erreurs de 1.
AugmenteDeplacements()	Incrémente m_nb_mouvements de 1.
BOOL JoueEncore()	Retourne m_joue.
BOOL SabotVide()	Utilisé pour le contrôle des règles de fin de jeu. Teste s'il y a un jeton dans le sabot ou dans la zone d'attente du joueur courant.
SupprimeSabot()	Envoie les messages suivants au sabot : "SupprimeJoueur(this) et ensuite NbJoueurs(). Si le nombre de joueurs propriétaires du sabot est 0, le sabot est détruit.
SupprimeResultat()	idem supprime sabot
SupprimeAttente()	idem supprime sabot
string QuelNom()	retourne le nom du joueur

10.14. La classe caractéristique

Les structures de données

Variable membre	Signification
TableauString m_liste_caract	Tableau de strings contenant les caractéristiques.

Les services offerts

Fonction	Objectif
Ajoute_caract(string)	Ajout d'une caractéristique au tableau de caractéristiques.
int NbCaract()	Retourne le nombre d'éléments du tableau de caractéristiques.
string Caract(int i)	Retourne l'élément d'indice i du tableau de caractéristiques.

10.15. La classe case

Les constantes utilisées

VUE : vue_concepteur, vue_joueur

TYPE_ECHIQUIER : grille, sabot, attente, resultat

Les structures de données

Variable membre	Signification
int m_n_ligne	Numéro de la ligne dans laquelle se trouve la case.
int m_n_col	Numéro de la colonne dans laquelle se trouve la case.
Echiquier* m_pt_parent	Pointeur sur l'échiquier propriétaire de la case.
Liste Pointeur m_jetons_dans_case	Liste de pointeurs sur les jetons se trouvant dans la case.
int m_pt_jeton_visible	Pointe sur un élément de m_jetons_dans_case. Le jeton d'indice m_pt_jeton_visible est le jeton actuellement visible de la case, c'est-à-dire le jeton que la case va afficher, le jeton qui sera sélectionné lors d'un clic dans la case.

Les services offerts

Fonction	Objectif
TYPE_ECHIQUIER QuelTypePere()	Retourne le type de l'échiquier propriétaire de la case.
BOOL ContientJetonSuperposable()	Indique si la case contient un jeton superposable.
BOOL Deplacement (jeton*, case*)	Demande à la case d'effectuer un déplacement. La case envoie le message EnlèveJeton(jeton*) à la case d'origine (case fournie en paramètre) et ensuite elle lance la fonction AjouteJeton(jeton*).
Echiquier* QuelPere()	Retourne un pointeur sur le propriétaire de la case.

Case(int, int, Echiquier*)	Constructeur de case. Il faut fournir comme paramètres : le numéro de ligne, le numéro de colonne et un pointeur sur l'échiquier propriétaire.
ChangeLigne(int)	Change le numéro de ligne d'une case.
ChangeCol(int)	Change le numéro de colonne d'une case
Inverse(View*, VUE)	Demande d'afficher en vidéo inverse le contenu d'une case dans une fenêtre. VUE indique la manière de calculer les coordonnées de la case à afficher : par rapport au coin supérieur gauche de la fenêtre pour la vue concepteur et par rapport à la position m_debut de l'échiquier pour la vue joueur.
Dessine(View*, VUE)	Dessine la case dans la fenêtre pointée.
AjouterJeton(jeton*)	Ajout d'un jeton dans la case, ce jeton est ajouté à la liste m_jetons_dans_case et devient le jeton visible.
EnleveJeton(Jeton* pJeton)	Enlève le jeton pJeton de la liste des jetons.
EnleveTousLesJetons()	Vide la liste m_jetons_dans_case.
EnleveJetonVisible()	Enlève le jeton visible.
int NbJetonsCase()	Retourne le nombre de jetons se trouvant dans la case, soit le nombre d'éléments de m_jetons_dans_case.
CJeton* JetonVisible()	Retourne un pointeur sur le jeton visible de la case.
CJeton* JetonCase(int i)	Retourne le ième pointeur de la liste m_jetons_dans_case.
CJeton* ChoisirJeton()	Boîte de dialogue permettant de choisir un jeton parmi ceux se trouvant dans la case.
JetonVisibleSuivant()	Demande d'afficher le jeton suivant de la case. (incrément de m_pt_visible).

10.16. La classe case de grille

Les structures de données

Variable membre	Signification
int m_pt_zone	Il s'agit d'un pointeur sur la liste m_zone. La zone m_zone(m_pt_zone) est la zone dans laquelle se trouve la case.
BOOL m_table_dep_defini	Indique qu'une table de déplacement a été définie pour la case.
BOOL m_hg	Indique si un jeton peut se déplacer d'une case vers le haut à gauche.
BOOL m_h	Indique si un jeton peut se déplacer d'une case vers le haut.
BOOL m_hd	Indique si un jeton peut se déplacer d'une case vers le haut à droite.
BOOL m_g	Indique si un jeton peut se déplacer d'une case vers la gauche.
BOOL m_c	Indique si la case peut être désignée comme case de destination d'un déplacement.

BOOL m_d	Indique si un jeton peut se déplacer d'une case vers la droite.
BOOL m_bg	Indique si un jeton peut se déplacer d'une case vers le bas à gauche.
BOOL m_b	Indique si un jeton peut se déplacer d'une case vers le bas.
BOOL m_bd	Indique si un jeton peut se déplacer d'une case vers le bas à droite.

Les services offerts

Fonction	Objectif
CaseDansZone(int)	Change l'indice de la liste des zones : m_pt_zone
int QuelleZone()	Retourne m_pt_zone.

10.17. La classe zone

Les structures de données

Variable membre	Signification
string m_nom_zone	Le nom donné à la zone.
Liste Pointeur m_cases_zone	Liste de pointeurs sur les cases de la zone.
TableauEntier m_caract_zone	Tableau des caractéristiques de la zone.
TableauEntier m_caract_concepteur_zone	Tableau des caractéristiques données par le concepteur à la zone.

Remarque : Pour la zone, une caractéristique est vue comme un entier. Lorsqu'il y a comparaison de caractéristiques, il y a comparaison entre entiers. Chaque entier est un indice du tableau des caractéristiques.

Les services offerts

Fonction	Objectif
TableauEntier* QuelleCaractZone()	Retourne un pointeur sur les caractéristiques de la zone.
AjouteCase(CaseGr*)	Ajoute un pointeur sur une case à m_cases_zone.
SupprimeCase(CCaseGr* pt_case)	Enlève le pointeur pt_case de m_cases_zone.
int NbCasesZone()	Retourne le nombre de cases de la zone ou le nombre de pointeurs contenus dans m_cases_zones.
LibererCases()	Vide la liste des pointeurs sur les cases de la zone et pour chaque case enlevée, envoie le message :

	CaseDansZone(0) pour mettre ces cases dans la zone "Reste de la Grille".
CaseGr* Case(int i)	Retourne un pointeur sur la ième case de la zone.
AjouteCaractConcepteur(int)	Ajout d'une caractéristique au tableau des caractéristiques concepteur.
AjouteCaractJeton(Jetons* pListeJetons);	Ajout de pointeurs au tableau des caractéristiques, les caractéristiques ajoutées sont celles qui sont contenues dans le jeton donné en paramètre.
EnleveCaractJeton(Jetons*)	Enlève du tableau des caractéristiques les caractéristiques d'un jeton. Elles ne sont enlevées que si elles ne sont contenues dans aucun autre jeton de la zone et si elles ne figurent pas dans le tableau des caractéristiques concepteur.
AjouteCaractZone(int)	Ajout d'une caractéristique au tableau des caractéristiques.
EnleveCaractZone(int)	Enlève une caractéristique du tableau des caractéristiques.
int NbCaractZone()	Retourne la taille du tableau des caractéristiques.
int NbCaractConcepteur()	Retourne la taille du tableau des caractéristiques concepteur.
int CaractZone(int i)	Retourne la ième caractéristique de la zone.
int CaractZoneConcepteur(int i)	Retourne la ième caractéristique donnée par le concepteur à la zone.
BOOL CaractDansConcepteur(int car)	Teste si la caractéristique "car" se trouve dans le tableau concepteur.
BOOL CaractDansZone(int car)	Teste si la caractéristique "car" se trouve dans le tableau des caractéristiques.
static CopierZones(Grille*, Grille*)	Création et copie de toutes les zones de la grille source vers la grille destination.

11. L'interface

Montrons par un exemple comment utiliser le générateur. Dans cette partie, je vais alterner les explications sur l'interface et les actions à exécuter pour créer les jeux. Pour distinguer rapidement entre les explications de l'interface et les actions, chaque fois que j'invite le lecteur à effectuer une action, je vais effectuer un décalage et écrire les actions à effectuer en italique.

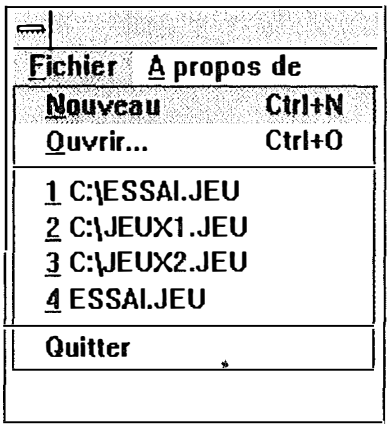
11.1. Le "jeu de mémoire"

Au démarrage du programme, le menu fichier repris ci-dessous est affiché.

Le menu fichier

- Le menu déroulant "*Fichier*" : permet la création d'un nouveau jeu, l'ouverture d'un jeu existant, d'ouvrir un fichier parmi les 4 derniers sauvés et de quitter le générateur.

- Le menu déroulant "*A propos de*" affiche une boîte de dialogue contenant des informations sur le générateur.



Choisissons l'item "Nouveau" du menu fichier pour créer un nouveau jeu.

Lorsqu'un nouveau jeu est créé, le programme affiche l'écran principal ainsi que la fenêtre de visualisation du jeu en cours de construction. Cette fenêtre montre à tout moment au concepteur le jeu tel qu'il sera vu par le joueur.

L'écran principal

L'écran principal est composé du menu principal, de la barre d'outils et de l'aire d'affichage. Dans l'aire d'affichage se trouve la fenêtre de visualisation.

Le menu de l'écran principal

Pour le menu de l'écran principal on trouve une barre de menu déroulant comptant dans l'ordre les items "*Fichier*", "*Création*" et "*Jeu*".

L'item "*Fichier*" fournit un menu reprenant les fonctions concernant la gestion des fichiers de jeux. Il s'agit de :

- "Fermer " qui ferme le fichier en cours (sans sauvegarde)
- "Sauver" qui sauvegarde le jeu dans un nouveau fichier
- "Sauver comme" qui sauvegarde le jeu dans le fichier existant ou s'il s'agit d'un nouveau jeu demande le nom du fichier à sauver.
- "Quitter" qui permet de quitter le programme.

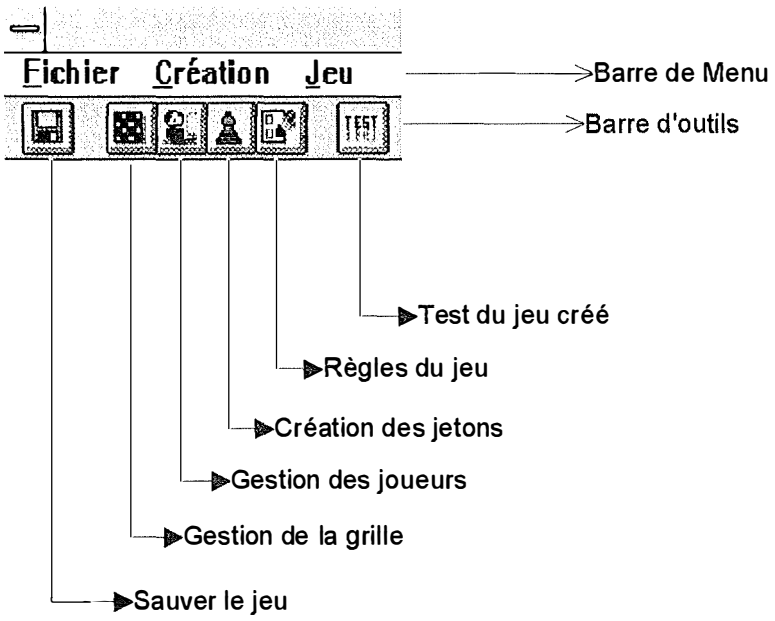
L'item "*Création*" fournit un menu composé des fonctions de gestion des différents éléments du programme. Il s'agit de :

- "Grille" qui ouvre la fenêtre de gestion de la grille
- "Joueur" qui ouvre la fenêtre de gestion des joueurs
- "Jetons" qui ouvre la fenêtre de gestion des jetons
- "Règles du jeu" qui affiche la boîte de dialogue des règles du jeu.

L'item "Jeu" fournit un menu reprenant l'item "Test", ce dernier permet de tester le jeu en cours de construction.

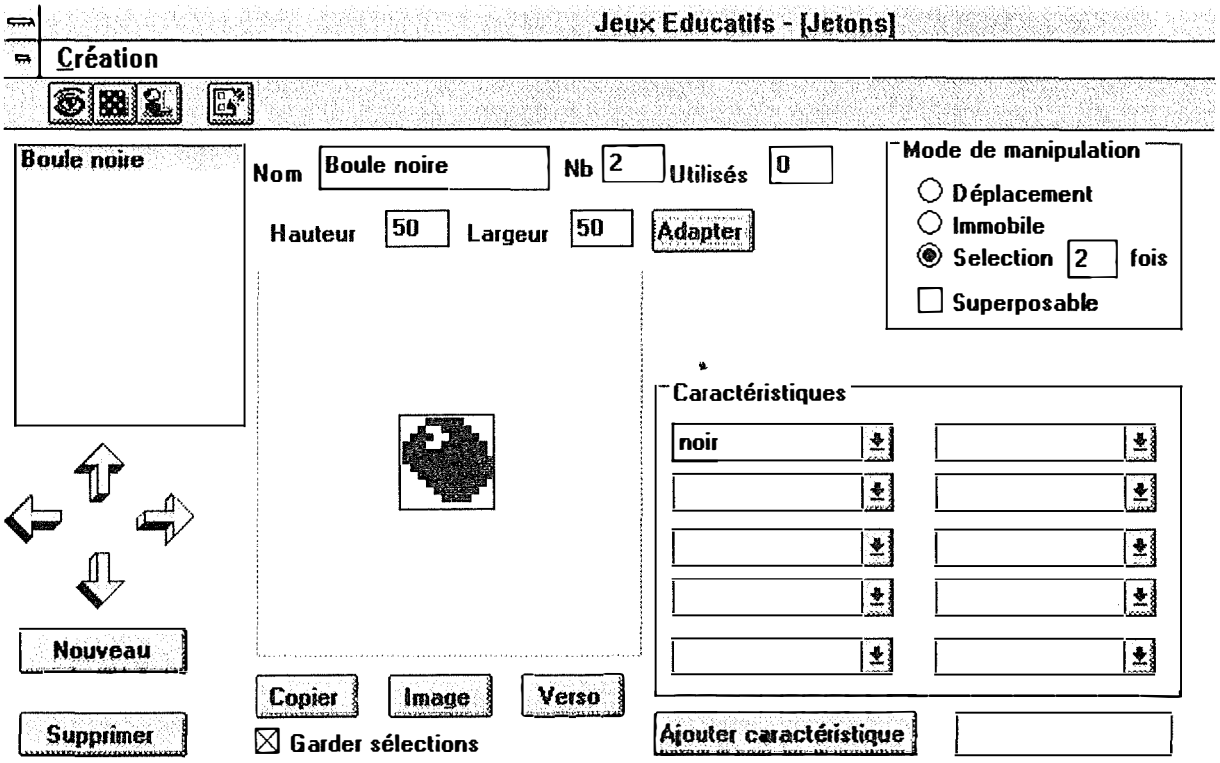
La barre d'outils de l'écran principal

La barre d'outils affiche des icônes permettant d'accéder rapidement à différents éléments du générateur.



11.2. Création des jetons

Cliquons sur l'icône "Création des jetons" de la barre d'outils de l'écran principal. Cela aura pour effet d'afficher la fenêtre de gestion des jetons.

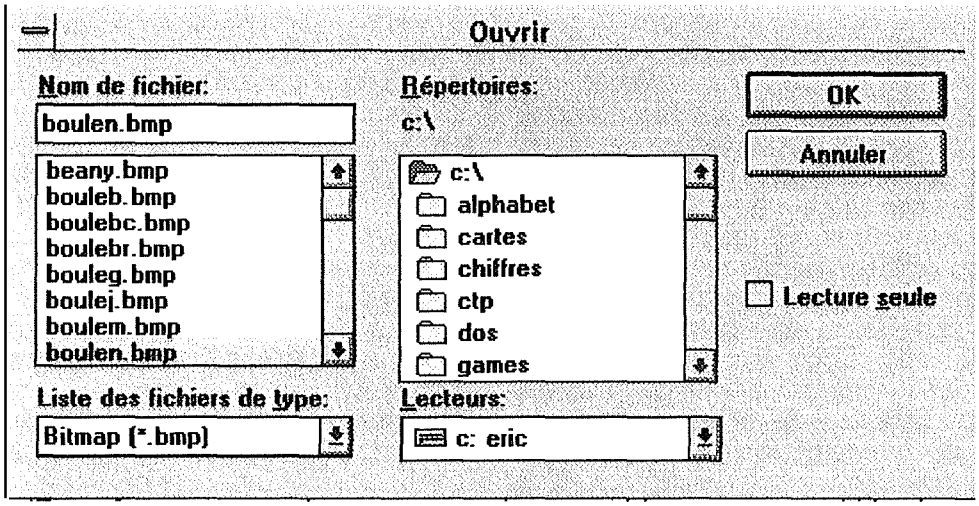


La barre d'outils de la fenêtre jetons, comporte différents icônes : l'oeil qui permet de revenir à la fenêtre de visualisation, la grille qui permet d'aller à la fenêtre de gestion de la

grille, le joueur qui permet d'aller à la fenêtre de gestion des joueurs et le formulaire qui affiche la boîte de dialogue règles du jeu.

Nnous allons premièrement introduire la taille des jetons. Pour ce faire, nous inscrivons 50 dans les cases hauteur et largeur et ensuite nous appuyons sur le bouton "adapter". Ce bouton va redessiner les jetons avec leur nouvelle taille.

Nous allons ensuite choisir le dessin du jeton en appuyant sur le bouton image, cela aura pour effet d'afficher la boîte de dialogue "ouvrir" qui permettra de choisir le fichier BMP contenant le dessin du jeton.



Un jeu de différents fichiers BMP est fourni avec le générateur, il s'agit des jetons contenant les lettres de l'alphabet, les chiffres et les opérateurs + - * / =, un jeu de carte et des boules de différentes couleurs. Pour le jeu de mémoire nous choisirons les boules de couleur.

Choisissons la boule noire en cliquant deux fois sur le fichier "boulen.bmp" et changeons le nom du jeton en "Boule noire".

Le mode de manipulation actuel du jeton est déplacement (voir le cadre "Mode de manipulation"). Dès lors, le bouton *Verso est grisé,* cela signifie qu'il ne peut pas être choisi.

Nous allons choisir sélection comme mode de manipulation ce qui entraîne l'activation du bouton "Verso". Nous allons cliquer sur "Verso" afin de choisir le dessin du dos du jeton.

Dans la case "Nb", nous allons inscrire 2 pour indiquer que nous désirons deux jetons "boule noire".

Il nous reste désormais à donner une caractéristique au jeton. Il convient préalablement de créer cette caractéristique.

Pour créer une caractéristique, il faut l'introduire dans la case "Ajouter caractéristique" et cliquer sur le bouton "Ajouter caractéristique". Une fois la caractéristique créée, nous pouvons la sélectionner pour notre jeton en cliquant dans une case du cadre "Caractéristiques".

Le jeton boule noire est créé.

Avant d'appuyer sur le bouton "Nouveau" pour créer un nouveau jeton, vérifions si la case à cocher "Garder sélections" est cochée.

La case à cocher "Garder sélections" permet de garder les options du jeton actuellement sélectionné lors de la création d'un nouveau jeton.

Création des jetons "boule verte", "boule vert clair", " boule rouge", "boule brune", "boule grise", "boule mauve" et "boule rouge".

Copions le premier jeton. Pour se positionner sur le premier jeton, il faut cliquer sur le bouton "flèche vers le haut". Cliquons ensuite sur le bouton "Copier" pour copier le premier jeton.

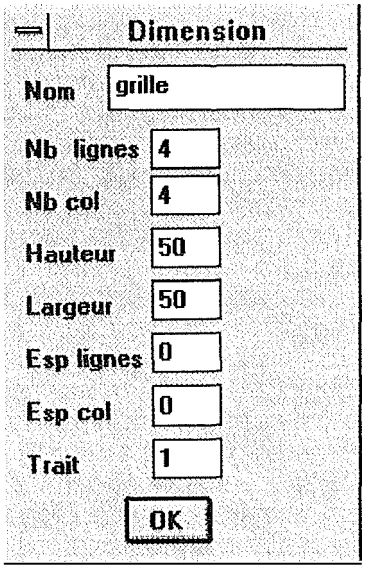
Nous allons ensuite créer la grille en cliquant sur le bouton "Grille" de la barre d'outils.

11.3. Création de la grille

Losque l'on demande l'affichage d'une fenêtre d'un échiquier (la grille, le sabot, la zone de résultat ou la zone d'attente), alors que cet échiquier n'a pas encore été créé, le programme affiche la boîte de dialogue de construction d'un échiquier.

La boîte de dialogue, Construction

Cette boîte de dialogue permet d'introduire les caractéristiques reprises à la figure ci-dessous :

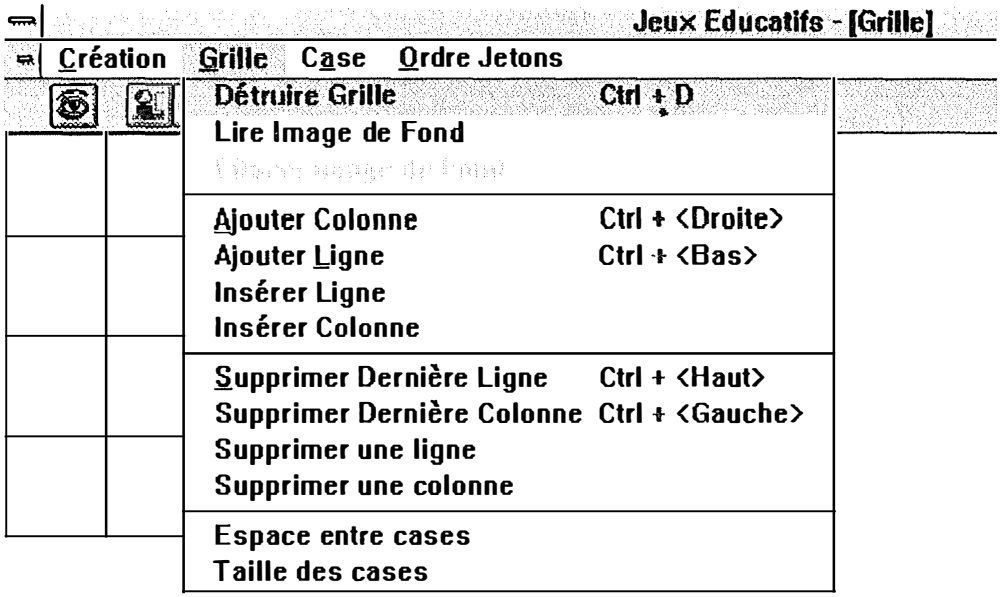


Pour le jeu de mémoire, nous allons créer une grille de 4 lignes et de 4 colonnes, nous allons introduire 4 dans les cases "Nb lignes" et "Nb col" et ensuite cliquer sur le bouton "OK".

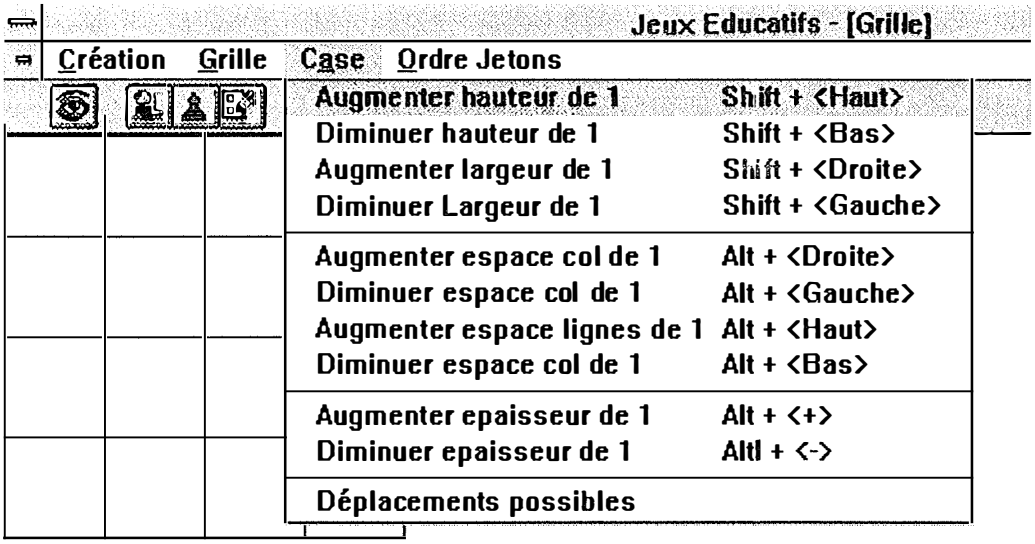
Une fois la grille créée, elle est affichée dans sa propre fenêtre. La fenêtre grille possède sa propre barre d'outils et sa propre barre de menu.

Le menu de la fenêtre grille

Le menu de la fenêtre grille, comporte une barre de menu déroulant comptant dans l'ordre les items *"Création"*, *"Grille"*, *"Case"* et "Ordre Jetons".



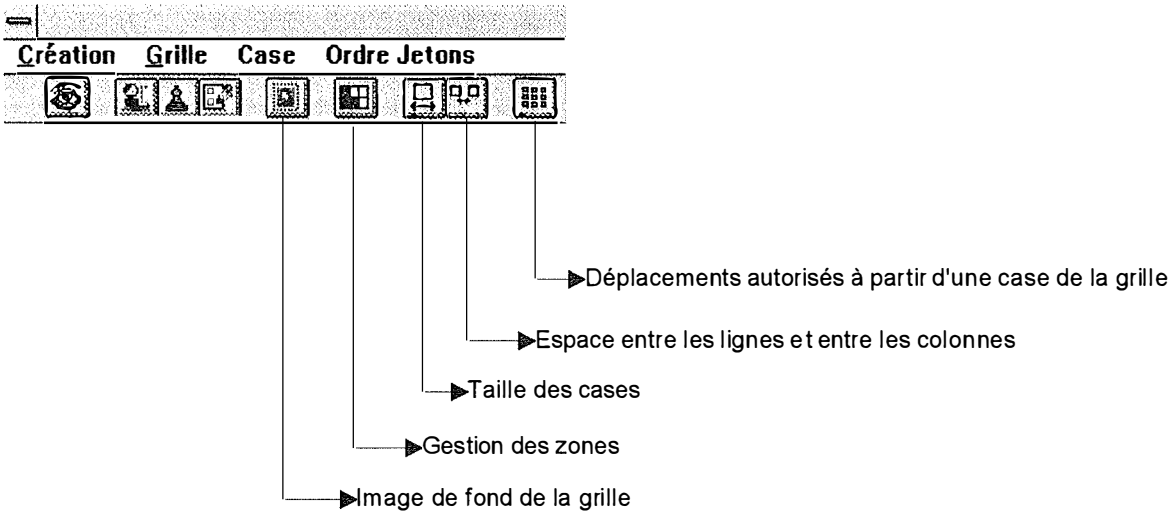
L'item "Grille" fournit un menu reprenant l'ensemble des fonctions de gestion de la grille offerte au concepteur. Il s'agit de la destruction de la grille, de l'ajout d'une colonne ou d'une ligne, de l'insertion d'une colonne ou d'une ligne, de la suppression de la dernière colonne ou de la dernière ligne, de la suppression d'une colonne ou d'une ligne, de déterminer l'espace entre les cases et de déterminer la taille des cases. Certaines de ces fonctions sont accessibles par des raccourcis claviers : il est possible d'ajouter une colonne avec la combinaison de touches : "Ctrl + <Droite>", ... (voir le menu ci-dessus).



L'item "case" fournit un menu reprenant l'ensemble des fonctions permettant de changer la taille des cases, l'espace entre les cases et l'épaisseur du trait utilisé pour dessiner les cases. A chacune de ces fonctions est associé un raccourci clavier.

L'item "ordre jetons" fournit un menu permettant au concepteur de choisir l'ordre dans lequel les jetons doivent être affichés lors du jeu. "Ordre aléatoire" indique que les jetons doivent être mélangés lors de chaque nouveau jeu, "Garder ordre" indique que les jetons doivent toujours se trouver à la même place lors de chaque nouveau jeu.

La barre d'outils de la fenêtre grille.



11.4. Coller un jeton dans la grille

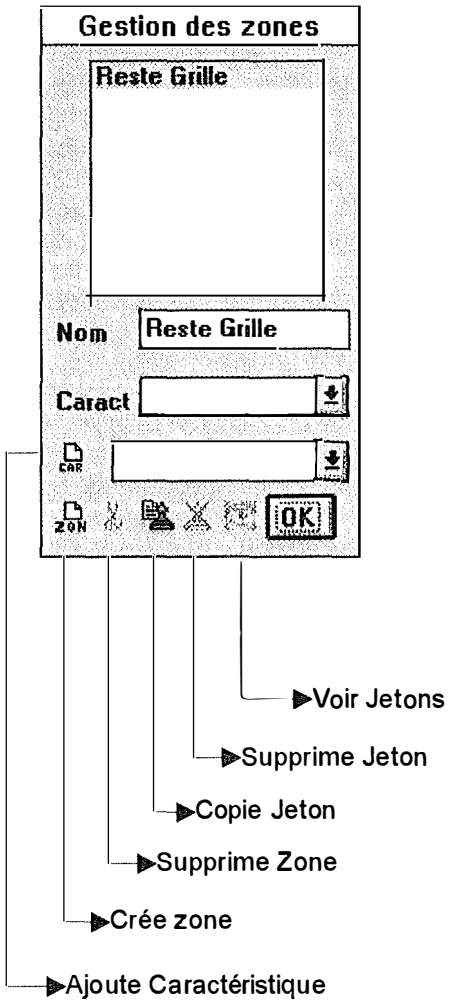
Pour pouvoir copier un jeton dans la grille il faut passer par la boîte de dialogue de gestion des zones.

Cliquons sur l'icône "Gestion des zones" de la barre d'outils de la fenêtre de la grille. .

La boîte de dialogue "Gestion des zones"

Comme nous pouvons le voir à la figure ci-dessus, il existe déjà une zone "Reste Grille". Cette zone a été créée en même temps que la grille et contient toutes les cases de celle-ci. Lorsque l'on sélectionne une zone, toutes les cases de la grille contenues dans cette zone sont affichées en vidéo inverse, c'est pourquoi toutes les cases de la grille sont affichées en vidéo inverse. La zone "Reste Grille" est une zone spéciale qui ne peut être supprimée et dont le nom ne peut pas être modifié.

Signification des icônes de la boîte de dialogue :



- **L'icône "Ajoute Caractéristique"** : Il permet d'ajouter une "caractéristique concepteur" à la zone courante. Il faut d'abord choisir cette caractéristique au moyen de la zone déroulante.

- **L'icône "Crée Zone"** : crée une nouvelle zone et rend cette zone courante.

- **L'icône "Supprime Zone"** : supprime la zone courante.

- **L'icône "Colle Jeton"** : permet de coller dans une case de la grille le jeton copié dans la fenêtre de gestion des jetons. Une fois un jeton copié il peut être collé tant que le nombre de jetons utilisés est inférieur ou égal au nombre de jetons.

- **L'icône "Supprime Jeton" :** enlève un jeton de la grille.

- **L'icône "Voir Jetons"** : lorsqu'un jeton dont le mode de manipulation est sélection est copié dans la grille, il est dessiné face vers la grille, seul son dos est visible. Lorsque l'on clique sur l'icône "Voir Jetons", la grille affiche tous les jetons face visible.

Cliquons sur l'icône "Colle Jeton" pour coller les deux jetons copiés.

Lorsque l'on clique sur l'icône "Colle Jeton", le curseur de la souris est changé et prend la forme de l'icône "colle jeton" pour indiquer qu'il faut sélectionner une case de la grille. Lorsque la case est sélectionnée, le jeton est collé.

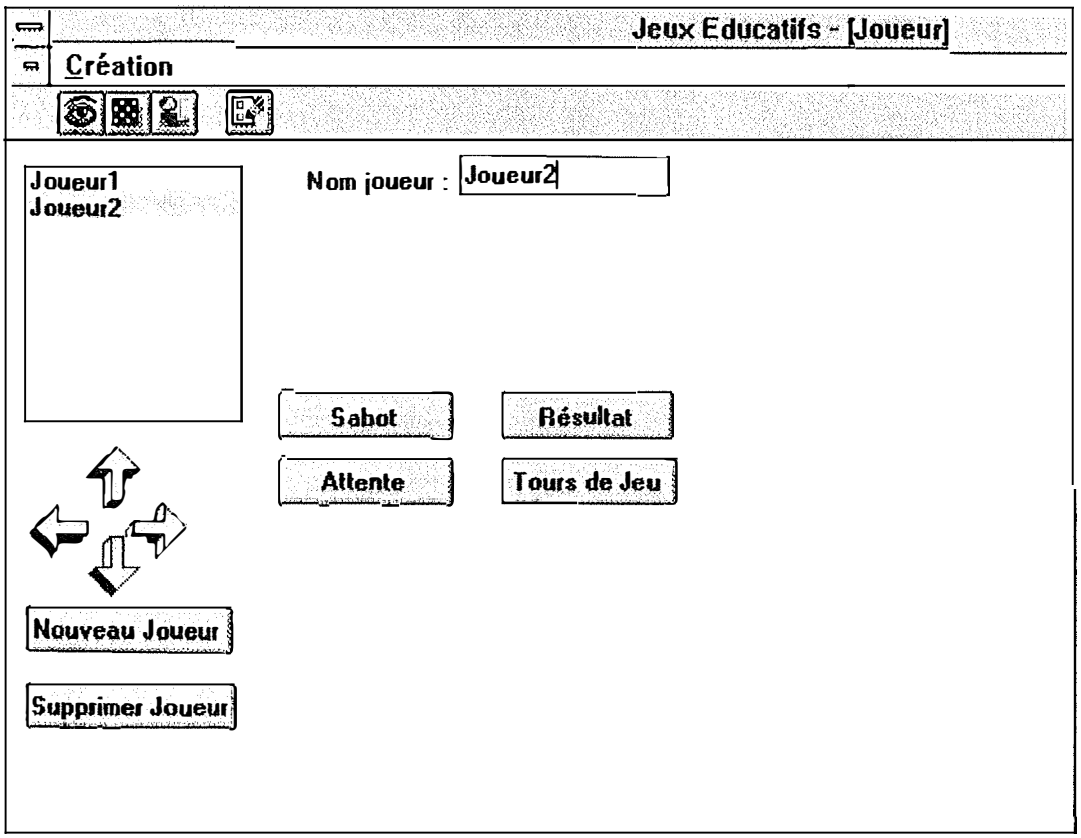
Pour coller les jetons suivants nous revenons à la fenêtre de gestion des jetons, sélectionnons le jeton suivant en appuyant sur le bouton "➔" et copions le jeton en appuyant sur le bouton "Copier". Revenons à la fenêtre grille pour y coller les jetons, et ainsi de suite pour tous les jetons.

Pour que le jeu ait un intérêt, il est souhaitable qu'à chaque nouvelle partie les jetons occupent une place différente dans la grille.

Vérifions si l'item Ordre Jetons de la barre de menu de la grille est positionné sur "Ordre Aléatoire" (option par défaut).

Le jeu de mémoire se joue à deux joueurs, pour créer les joueurs, cliquons sur l'icône joueur de la barre d'outils.

11.5. Création des joueurs

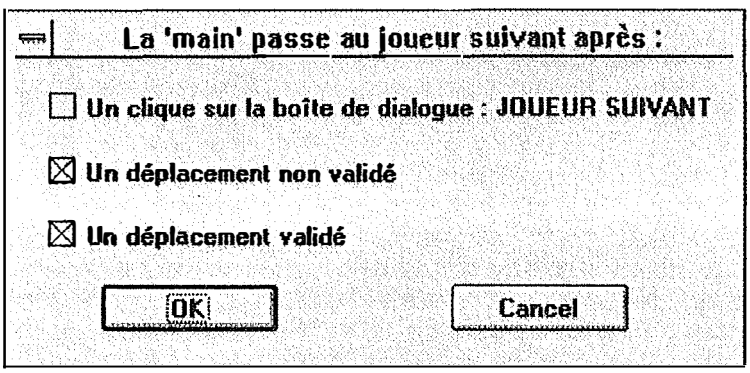


Le "jeu de mémoire" se joue à deux joueurs.

Le nom du joueur peut être changé en cliquant dans la case affichant le nom. Pour créer un deuxième joueur il faut cliquer sur le bouton "Nouveau Joueur", nous pouvons changer son nom en "Joueur2".

Pendant la partie, le nom du joueur ayant la main est affiché dans la zone d'affichage située sur la dernière ligne de l'écran.

Déterminons à quel moment la main change, pour cela il faut cliquer sur le bouton "Tours de Jeu".



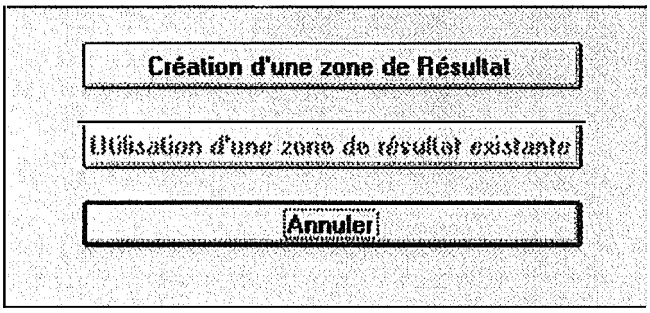
Les options par défaut conviennent pour le jeu de mémoire, cliquons sur le bouton "Cancel" pour fermer la boîte de dialogue.

11.6. Création de la zone de résultat

Pour ajouter de l'intérêt au jeu, il est utile de créer deux zones de résultat, une pour chaque joueur.

Sélectionnons d'abord le premier joueur et cliquons sur le bouton "Résultat".

La boîte de dialogue suivante est affichée.



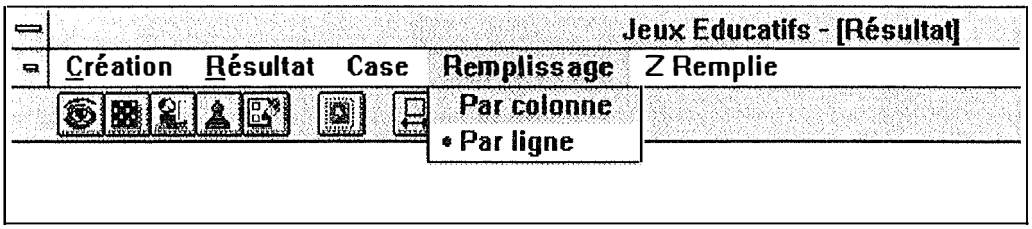
Remarquons que le bouton "Utilisation d'une zone de résultat existante" n'est pas autorisé car il n'y a pas encore de zone de résultat créée.

Cliquons sur le bouton "Création d'une nouvelle zone de résultat" pour faire apparaître la boîte de dialogue de création de la zone de résultat.

Une fois la zone de résultat créée, la fenêtre de gestion de la zone de résultat est affichée. La barre d'outils de la zone de résultat possède en plus de celle de la grille l 'icône "Grille".

Le menu de la fenêtre de résultat

La barre de menu de la zone de résultat est composée en plus des items "Création", "Résultat" et "Case" des items "Remplissage" et "Z Remplie".



L'item "Remplissage" permet de déterminer l'odre par lequel les jetons arrivant dans la zone de résultat sont affichés : ligne ou colonne par colonne.

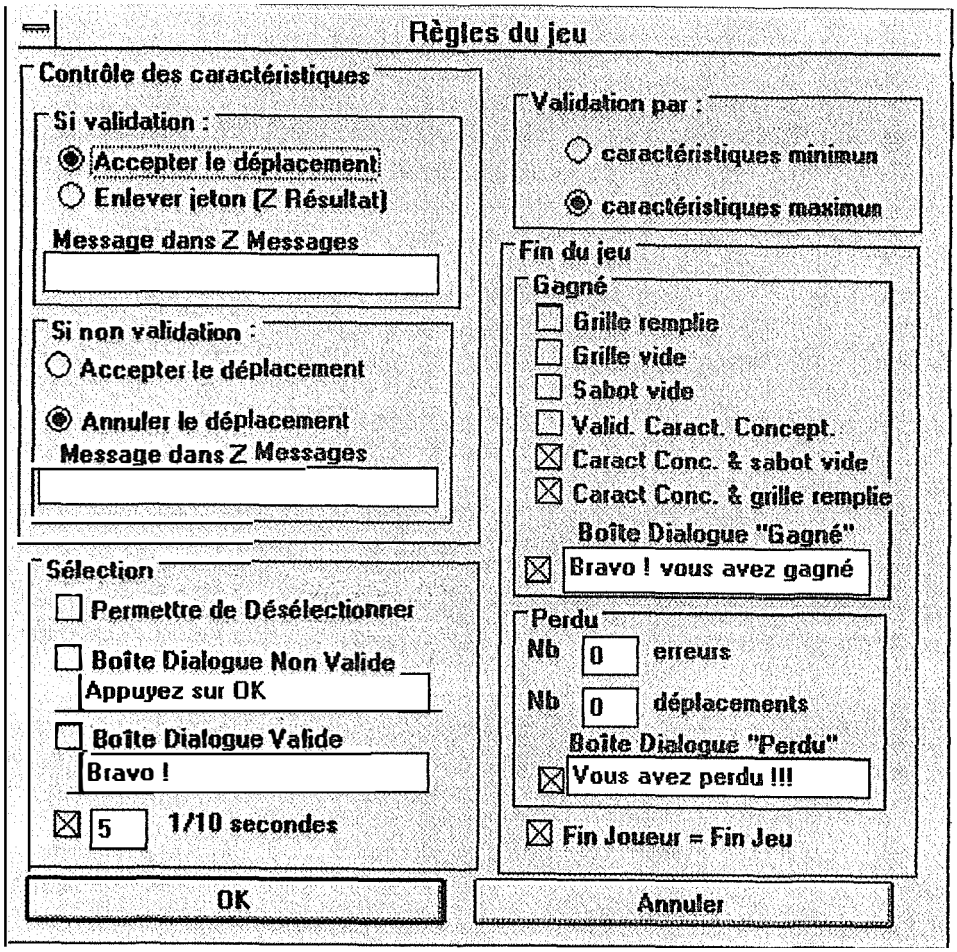
L'item "Z Remplie" permet de déterminer ce qu'il faut faire lorsque la zone de résultat est remplie et lorsque un nouveau jeton doit y être affiché. Les options possibles sont :

- Décaler Ligne (enlève les jetons de la première ligne et recopie tous les jetons une ligne plus haut)
- Décaler Colonne (enlève tous les jetons de la première colonne et recopie tous les jetons une colonne vers la gauche),
- Vide (enlève tous les jetons)
- Rien (arrêt d'affichage des jetons dans la zone de résultat).

Pour terminer la création du jeu voyons les rèlges du jeu.

Nous allons cliquer sur le bouton "Formulaire" de la barre d'outils qui permet de faire apparaître la boîte de dialogue "Règles du Jeu".

Modification des règles du jeu



Toutes les options par défaut des règles du jeu sont déjà cochées. Il suffira d'adapter ces règles au jeu de mémoire.

- Cadre "Déplacements dans la grille", ce cadre n'est pas utilisé pour le jeu de mémoire car le mode de manipulation des jetons est "sélection".

- Cadre "Sélection", les options par défaut de ce cadre ne doivent pas êtres changées.

- Cadre "Validation par", vu qu'il n'y a qu'une seule caractéristique par jeton, les validations maximum et minimum produisent le même résultat.

- Cadre "Fin du jeu", sous-cadre "Gagné",

Il faut changer les règles de fin de jeu en ne cochant que les cases "Grille vide" et boîte de dialogue.

- Cadre "Fin du jeu", sous-cadre "Perdu"

Changeons les nombres d'erreurs et de déplacements en 999.

Si un nombre d'erreurs maximum a été introduit dans le cadre perdu, à chaquenouveau tour de jeu, le programme va afficher le nombre actuel d'erreurs du joueur ayant la main. De même si un nombre maximum de déplacements a été introduit, lors de chaque tour de jeu, le programme affiche le nombre de déplacements déjà effectué par le joueur courant.

Le "jeu de mémoire" est maintenant terminé.

11.7. Le jeu du taquet

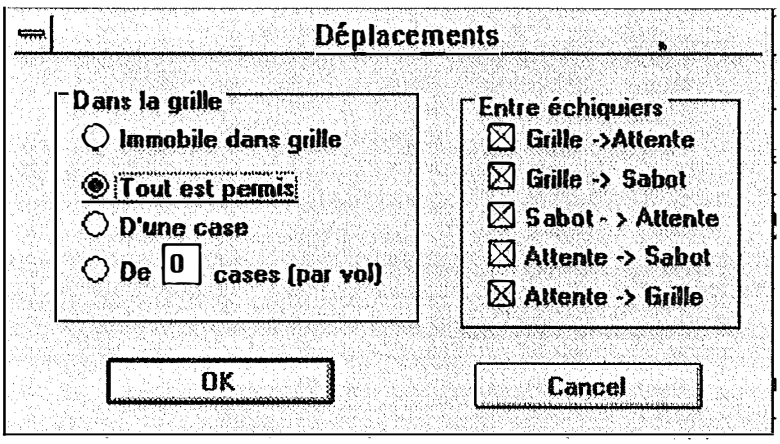
A l'aide d'un nouvel exemple montrons comment créer des zones, les tables de déplacement de la grille et les règles de déplacements des jetons.

Création des jetons

Nous commençons par créer 8 jetons. Cette fois nous allons utiliser les fichiers "1.bmp" jusque "8.bmp" pour les dessins des jetons. Ces fichiers représentent les chiffres de 1 à 8.

Nous allons d'abord ajouter la caractéristique "1" au jeton de nom "1" et ayant comme fichier de dessin "1.bmp", ..., "8" au jeton de nom "8" et ayant "8.bmp" comme fichier de dessin. Les jetons sont "non superposable".

Les jetons pouvant se déplacer dans la grille, nous allons cliquer sur l'option Déplacement du mode de manipulation, cela entraîne l'affichage de la boîte de dialogue déplacement.



Changement de l'option du cadre "dans la grille" en "D'une case" et cliquons sur le bouton "OK" pour valider la modification.

Vu qu'il n'y a ni sabot, ni zone d'attente ni zone de résultat, nous ne nous interressons pas au cadre de droite "Déplacements entre Echiquiers.

Création de la grille

Créons ensuite une grille de 3 lignes sur 3 colonnes.

Création des zones de la grille.

Pour pouvoir détecter la fin du jeu, nous devons indiquer au programme pour chaque case de la grille quel jeton cette case doit posséder. Pour ce faire, nous allons créer autant de zones qu'il y a de cases dans la grille.

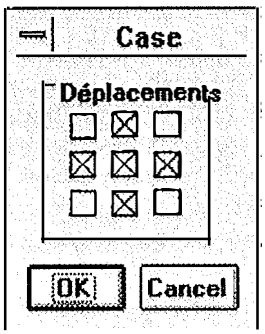
Cliquons sur le bouton "Gestion des zones" de la fenêtre grille et collons les jetons dans la grille.

Pour créer une nouvelle zone, il faut cliquer sur l'icône "Crée zone". Changeons le nom de cette zone en "1". Pour ajouter la caractéristique "1" déroulons la liste des caractéristiques et choisissons dans la caractéristique "1". La caractéristique "1" est maintenant sélectionnée, elle sera rajoutée à la zone en cliquant sur le bouton "Ajoute Caractéristique". Cliquons sur la case de la première ligne et première colonne pour la rajouter dans la zone. Faisons de même pour les zones "2" ... "8".

Remarque, il n'y a pas besoin de créer la zone "9" car nous n'avons créé que 8 jetons. La dernière case de la grille sera contenue dans la zone "Reste Grille".

Modification de la table des déplacements des cases

La table des déplacements par défaut de la grille ne doit pas être modifiée. Par contre pour interdire à un jeton de traverser la grille d'un bord à l'autre, il va falloir changer les tables de déplacements de chaque case se trouvant au bord de la grille.



Cliquons deux fois sur la case située dans le coin supérieur gauche pour modifier la table de déplacements de cette case.

Les "x" représentent les directions autorisées pour un déplacement d'une case.

Un jeton se trouvant dans cette case ne pourra pas aller vers le haut ni vers la gauche, nous allons donc enlever les croix se trouvant dans ces directions.

Cliquons sur toutes les autres cases du bord de la grille pour changer les tables de déplacements de ces cases.

Modifications des règles du jeu

- Cadre "Déplacement dans grille"
    - sous cadre "Caractéristiques Validées", il faut accepter un déplacement dans la grille même lorsque les caractéristiques sont validées.
    - sous cadre "Caractéristiques non validées", il faut accepter également ce déplacement.

    Modifier le sous cadre "Caractéristiques non validées" en acceptant le déplacement non valide.

- Cadre "Sélection", il n'y a pas de jeton dont le mode de manipulation est "sélection".
- Validation par : idem que pour le "jeu de mémoire", il n'y a qu'une seule caractéristique par jeton, cela n'a donc pas d'effet de changer la validation.
- Fin du jeu
    - sous cadre "Gagné". Uniquement lorsque les caractéristiques concepteurs sont validées.

    Modification des règles de fin de jeu "Gagné" en "Valid. Caract. Concept.".

    - sous cadre "Perdu". On peut comme pour le "jeu de mémoire" introduire 999 comme nombre maximum d'erreurs et de déplacements.

11.8. Pigeon Vole

Création des jetons

Il s'agit de déterminer pour une série d'animaux s'il s'agit d'un animal volant ou non. Pour ce jeu, nous allons créer une série de jetons. Chaque jeton a un dessin d'animal, il a comme mode de manipulation : "déplacement". Le type de déplacement dans la grille est "Tout Permis". Les caractéristiques "volant" et "non volant" doivent être crées et chaque jeton doit avoir une de ces deux caractéristiques.

Copions le premier jeton.

Création de la grille

La grille sera composée de deux lignes et d'un certain nombre de colonnes (ce nombre dépend du nombre de jetons créés).

La grille est découpée en deux zones. Chaque zone regroupant les cases d'une ligne. La première zone a comme caractéristique "volant" et la deuxième zone a comme caractéristique "non volant".

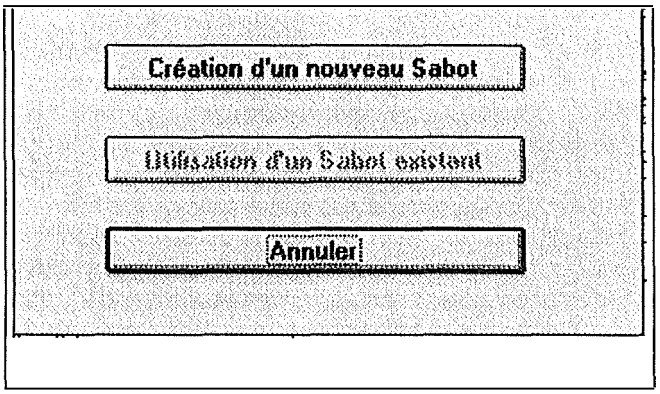
On va ensuite cliquer sur le bouton de gestion des zones et créer deux zones. La zone de nom "volant" pour laquelle on ajoute la caractéristique "volant" et composée des cases de la première ligne et la zone de nom "non volant" pour laquelle on ajoute la caractéristique "non volant" et composée des cases de la deuxième ligne.

Création du sabot

On va ensuite créer le sabot.

Il faut sélectionner le joueur pour lequel on désire ajouter un sabot (ici il n'y a qu'un seul joueur). Pour ce faire, il faut cliquer sur l'icône "Joueur" de la barre d'outils" et cliquer sur le bouton "sabot".

Lorsque l'on veut ajouter un sabot à un joueur, le programme demande s'il sagit d'un nouveau sabot ou d'un sabot existant.



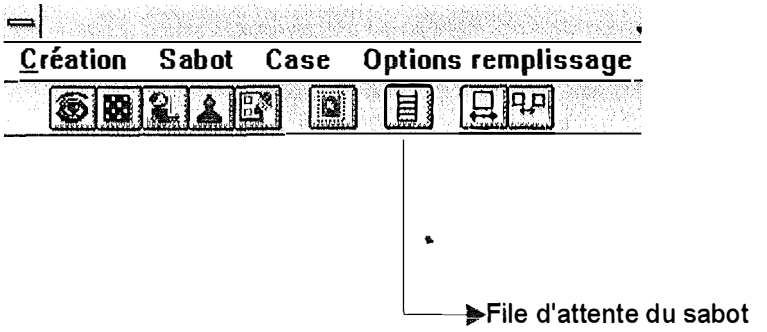
Pour créer le sabot, il faudra cliquer sur le bouton "Création d'un nouveau Sabot", ce qui provoque l'affichage de la boîte de construction du sabot.

La barre de menu du sabot

- **Les items "Création", "Sabot" et "Case"** sont les mêmes que pour la grille.

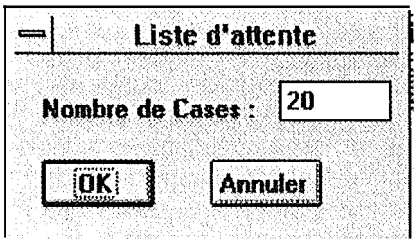
- **L'item "Options remplissage" :** indique à quel moment il faut remplir le sabot, à chaque fois qu'un jeton quitte le sabot, lorsqu'il est vide ou à la demande du joueur. Pour le jeu pigeon vole, nous allons accepter l'option par défaut : lorsqu'un jeton quitte le sabot.

La barre d'outils du sabot



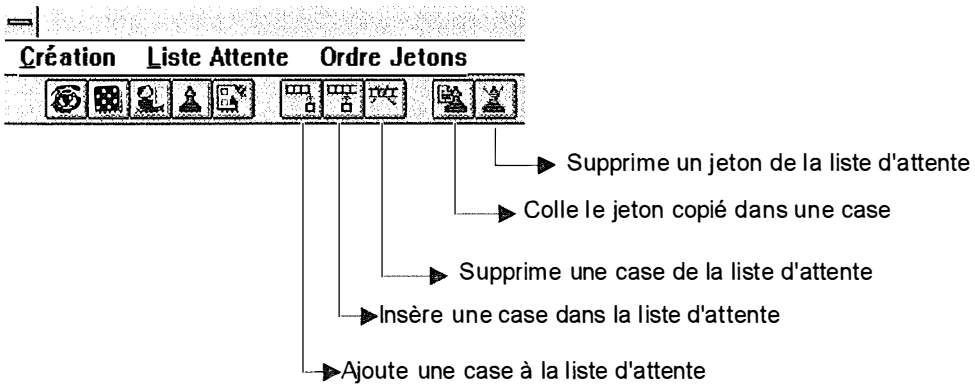
Cliquons sur le bouton "File d'attente du sabot" pour coller les jetons dans le sabot.

La file d'attente du sabot



Lorsque l'on accède pour la première fois à la file d'attente du sabot, il y a affichage de la boîte de dialogue de création du sabot. Nous déterminons le nombre de cases de la file d'attente.

Ensuite, il y a affichage de la fenêtre "file d'attente" du sabot. Nous allons copier les jetons de la fenêtre "gestion des jetons" et les coller dans les cases de la file d'attente.



La barre de menu de la file d'attente

- **L'item "Liste Attente" :** permet d'ajouter une case en fin de liste, d'insérer une case ou de supprimer une case, ainsi que de détruire la liste.

- **L'item "Ordre Jeton"** : permet de copier les jetons de la file d'attente vers le sabot en gardant l'ordre des jetons ou de mélanger les jetons (par défaut, il y a mélange des jetons).

Les règles du jeu

Il y a deux options pour les règles du jeu :

- On accepte les jetons non validés et on enlève les jetons validés.

- On refuse les jetons non validés et on accepte les jetons validés.

Le jeu est fini lorsque le sabot est vide et que les caractéristiques concepteur sont validées.

12. Conclusion

12.1. Rappel de l'objectif

Je m'étais fixé comme objectif d'implémenter un générateur de jeux destiné à être utilisé par des éducateurs s'occupant de personnes présentant un handicap mental léger. Le but du générateur vise à permettre à ces éducateurs de concevoir des jeux éducatifs.

Il fallait un programme simple à utiliser mais permettant de créer des jeux suffisamment différents l'un de l'autre de manière à renouveler l'intérêt des joueurs.

12.2. Ce que j'ai réalisé

L'analyse

J'ai pris comme point de départ l'analyse effectuée par Yves Lowette et Jean-Christophe Marchal et en particulier le formulaire de définition des règles du jeu.

Dans une première phase, j'ai élargi les familles de jeux pouvant être générées par le générateur en introduisant de nouveaux concepts. Ces nouveaux concepts sont :

- le joueur. Un jeu peut être multi-joueurs (c'est-à-dire que le nombre de joueurs est quelconque) ou bien avec un nombre de joueurs déterminé.
- le tour de jeu. Il s'agit de déterminer à quel moment la main passe d'un joueur au joueur suivant.
- les différents types de déplacements des jetons à l'intérieur de la grille. Un jeton peut demeurer immobile dans la grille ou se déplacer dans toute la grille, peut se déplacer d'une case ou de x cases.
- le dé. Ce concept a été analysé mais pas implémenté.

Dans une seconde phase, j'ai essayé de simplifier le formulaire proposé par Lowette et Marchal. Les simplifications apportées au formulaire de définition des règles du jeu sont les suivantes :

- Les règles de validation retenues sont : validation par caractéristiques maximum ou par caractéristiques minimum. L'option par défaut étant la validation par caractéristiques maximum.

- déplacement du concept "règles de déplacements entre échiquiers" du formulaire vers l'interface de définition du jeton.

J'ai enfin défini un nouveau formulaire de définition des règles du jeu.

La conception

L'analyse réalisée, j'ai entamé la partie purement informatique du travail, consistant à élaborer le générateur de jeux.

J'ai identifié tout d'abord une série d'objets. J'ai ensuite expliqué les liens existants entre ces objets et les fonctions offertes par ces objets.

L'interface

Dans cette partie, j'explique comment utiliser le logiciel à travers trois exemples. Grâce à ces exemples, les différentes fenêtres et menus de l'interface sont explicités.

Prolongement du travail

Le présent logiciel pourrait être complété par l'implémentation des concepts suivants : dé, pièce et des caractéristiques liées à ces concepts (pivotable, ...).

13. Table des matières