THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Réalisation d'un simulateur de banc d'optique didactique

Weemaes, Goswin

Award date:
1998

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX, NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

Réalisation d'un simulateur de banc d'optique didactique

Goswin Weemaes

Mémoire présenté en vue de l'obtention du grade de

Licencié en Informatique

Année Académique 1997 - 1998

Summary

Today, educational software plays a more and more important role in teaching. The advantages of using this kind of programs at school are being highlighted. Particularly, they involve pupils in a more active way.
The goal of this thesis was to create an optic bench simulator. The use of scientific experiment simulators will never replace the practice, however, they allow a large range of possibilities that are not always really feasible. In the case of our program, pupils can see virtual images, define a lens the way they wish it, ...

But we wanted to take a step further the classical simulator, and the special feature of our simulator is that it gives the opportunity to make the link between the experimental phenomenons and the equations governing them. This way, pupils can as well observe as understand what happens on the optical bench.

Our software is written in object oriented (Delphi 2.0). This thesis was an opportunity for us to deepen our knowledge of this language, and particularly its graphical part.

Résumé

Aujourd'hui, les logiciels éducatifs jouent un rôle de plus en plus important. On commence à réaliser les avantages que présentent l'utilisation de ce genre de programmes dans le cadre actuel de l'enseignement : elle incite les élèves à interagir avec la matière.
Le but de ce mémoire était de réaliser un simulateur de banc d'optique. Les simulateurs d'expériences scientifiques, de façon générale, ne remplacent jamais les travaux pratiques, mais permettent tout de même de proposer des actions qui sont difficiles ou impossibles à réaliser dans la réalité. Dans le cas de notre logiciel, cela se traduit par la possibilité de visualiser des images virtuelles, de définir les lentilles dont on a besoin, ...

Mais nous avons voulu aller encore plus loin que la simple simulation : l'originalité de notre simulateur réside dans le fait qu'il permet de faire le lien entre les équations et les phénomènes optiques observés. Ainsi, l'élève peut observer *et* comprendre ce qui se passe sur le banc d'optique.

Le logiciel a été écrit en orientée-objet (Delphi 2.0), ce qui nous a permis d'approfondir nos connaissances en la matière (surtout son aspect graphique)

Remerciements

Au terme de ce mémoire, je tiens à remercier plusieurs personnes sans qui ce document n'aurait jamais vu le jour :

Monsieur le Professeur C. Cherton pour m'avoir accueilli si gentiment dans son bureau de nombreuses fois, pour m'avoir remis à certains moments sur le droit chemin grâce à ses conseils et ses remarques pertinentes.

Paloma ...

Mes parents, ma sœur et ma grand-mère, qui m'ont toujours suivi et soutenu tout au long de mes études.

Les étudiants de l'Institut d'Informatique avec qui j'ai pu vivre des moments formidables et inoubliables.

Mes amis sur qui j'ai pu compter dans les moments difficiles.

Table des matières

Introduction

Dans la plupart des écoles secondaires, les élèves reçoivent, dans le cadre des cours de physique, un certain nombre d'heures sur l'optique géométrique. On leur apprend différentes idées sur la lumière, sur la réflexion de la lumière, sur le phénomène de réfraction, sur les lentilles, ... On leur en explique le fonctionnement et l'utilité. Afin de faire interagir les élèves avec la matière, on leur donne des exercices a résoudre. Mais dans le cadre d'un cours comme celui-ci, ne serait-il pas plus approprié de leur proposer des exercices pratiques, pour qu'ils puissent expérimenter par eux-mêmes ? Ceci leur permettrait naturellement de mieux maîtriser la matière. Malheureusement, le matériel expérimental coûte cher et, de plus, est fragile.

C'est ici que les ordinateurs, et surtout les logiciels éducatifs qui les équipent, peuvent avoir un rôle important à jouer. Cela fait maintenant quelques années que les ordinateurs entrent petit à petit dans les écoles et dans les classes. En soi-même, l'ordinateur n'est pas utile, mais c'est grâce au logiciel qui l'accompagne que ce duo devient un vrai outil de travail pour les élèves. Les avantages apportés par les didacticiels sont multiples et dépendent du type de logiciel utilisés. Ils laissent une plus grande autonomie à l'élève qui peut évoluer à son propre rythme. L'acquisition des connaissances se fait de manière interactive. On a constaté que cette interaction incite les élèves à s'engager plus activement dans l'apprentissage et la réflexion que pendant les leçons traditionnelles.
Dans le cas qui nous intéresse, le cours d'optique géométrique en secondaire, nous désirons que les élèves puissent utiliser l'ordinateur pour réaliser des expériences optiques comme sur un vrai banc d'optique. Ceci nous amène donc à la réalisation d'un simulateur d'un banc d'optique. Le simulateur est un type de logiciel éducatifs parmi les autres. Il a pour but de modéliser le monde réel afin que l'élève puisse manipuler et étudier ce monde plus facilement. Dans le cadre de ce mémoire, nous n'avons pas l'intention de modéliser toutes les possibilités offertes par l'optique géométrique, mais nous nous limitons à l'étude des lentilles sphériques minces.

L'objectif premier est de construire un logiciel qui permette à un élève de réaliser des montages optiques comme sur un vrai banc d'optique. Un des grands avantages du simulateur par rapport à un banc traditionnel est qu'il a un éventail de possibilités quasi illimitées. Citons déjà comme exemple le fait que l'élève n'est pas restreint dans le choix de ses lentilles, mais qu'il peut définir ses propres lentilles en leur donnant les caractéristiques qu'il désire.
L'objectif suivant est d'augmenter l'intérêt pédagogique et l'originalité de notre simulateur en y intégrant un deuxième volet qui permette à l'élève de faire le lien entre les phénomènes physiques qu'il observe et les équations qu'il a vues au cours théorique. L'aspect "simulation" du logiciel va lui permettre de visualiser les évènements, alors que l'aspect "équations" va lui permettre de justifier et de comprendre les résultats obtenus.
Cette intégration nous mènera à proposer deux modes de fonctionnement du logiciel : un mode "libre" où l'élève prend toutes les initiatives pour explorer le monde qui lui est proposé par le programme, et un mode "problème" qui présente une approche plus directive, dans laquelle l'élève se base sur un exercice pour tester et approfondir ses connaissances.

Ce document a été divisé en cinq chapitres :

- Dans le premier chapitre nous décrivons les objectifs du projet. Nous situons ce travail dans le contexte des logiciels éducatifs et plus particulièrement dans celui des simulateurs.
- Le deuxième chapitre est un rappel des principes de l'optique géométrique. Nous nous limitons naturellement aux aspects qui interviennent dans notre logiciel.
- Ensuite, nous passons à l'analyse fonctionnelle. Puisque l'interface joue un rôle important dans les simulateurs, nous avons subdivisé ce chapitre en une analyse des fonctionnalités et une analyse de l'interface.
- Le chapitre cinq aborde le problème de l'analyse conceptuelle. Cette analyse nous permet d'aboutir à une architecture logique en niveaux.
- Le dernier chapitre est consacré à l'implémentation proprement dite du logiciel : comment avons nous concrétisé les fonctionnalités du programme ?

Chapitre 1

Description du projet

1.1 Les ordinateurs comme outils d'enseignement et d'apprentissage

On réalise depuis plusieurs années que l'ordinateur possède un sérieux potentiel de diversification et d'enrichissement de la relation pédagogique tant dans sa composante "enseignement" que dans son aspect "apprentissage".
Le rôle le plus classique de l'ordinateur [Duc94, Duc91] est celui d'outil d'aide à l'enseignement et à l'apprentissage. Ce rôle peut être vu de trois façons différentes :

- *L'ordinateur en tant que tuteur* : ce que l'on recherche dans ce cas, c'est que le logiciel agisse et réagisse comme le ferait un bon enseignant, ce qui donne lieu à un dialogue personnalisé entre l'élève et la machine.
- *L'ordinateur en tant que outil* : l'ordinateur va aider l'enseignant dans sa tâche en le libérant de certaines activités répétitives ou fastidieuses, pour qu'il ait plus de temps pour réaliser les activités pour lesquelles il est irremplaçables (remédiations, rattrapages, ...)
- *L'ordinateur en tant que mini-monde à explorer* : dans ce type de logiciel, l'élève se trouve plongé dans un "petit monde" qu'il va peu à peu explorer. Grâce à cette exploration (où ses actions engendrent des réactions de l'environnement), il va reconstruire dans sa tête le monde sur lequel il agit et les règles qui gouvernent celui-ci.

Il existe un nombre non négligeable de types différents de logiciels éducatifs [OCD89] : programme d'exercices, système tuteur intelligent, simulateur, résolution de problèmes, jeu pédagogique, ... Les rôles que remplissent chacuns de

ces types de logiciels ne sont naturellement pas toujours les mêmes, mais on peut dire qu'ils sont tous susceptibles d'apporter leur grain de sel dans la pédagogie et ceci à leur façon.

1.2 Les simulateurs

Avec ce type de logiciel, l'ordinateur sert à modéliser le monde réel ou un ensemble donné de conditions réelles de façon à pouvoir les manipuler et les étudier plus facilement. Les simulateurs peuvent couvrir un large éventail de phénomènes, depuis les mouvements des planètes jusqu'aux évènements historiques, en passant par les expériences scientifiques qui seraient trop dangereuses, trop coûteuses ou trop longues à réaliser dans un vrai laboratoire. L'utilisateur du simulateur peut tester des hypothèses en modifiant les variables du modèle et observer les effets de ces changements au sein de l'environnement de la simulation.

Un bon programme de simulation à but pédagogique réunit les caractéristiques suivantes [Swi] :

- Le programme de simulation doit être simple à utiliser, mais son objet doit être relativement complexe pour que l'utilisateur soit forcé de mener une recherche, qu'il soit poussé à se poser des questions, à effectuer des essais.
- Il doit placer l'élève dans une situation de travail aussi proche que possible des conditions réelles, exploitant à cet effet les possibilités étendues que présentent les ordinateurs actuels (graphisme, grande capacité d'interaction, son, vitesse de calcul, ...). L'utilisateur doit pouvoir poser des actes semblables à ceux qu'il accomplirait dans le monde réel.
- Les résultats de la simulation doivent lui être montrés, et non décrits. Ainsi, c'est l'élève qui doit observer et interpréter les résultats : un logiciel qui le ferait à sa place ne serait pas un bon logiciel didactique.
- Le phénomène simulé doit l'être avec suffisamment de détails et de réalisme, pour ne pas masquer la complexité du réel, dont on n'explore finalement qu'un modèle. Lors de la simulation, il faut que l'utilisateur soit confronté au problème du repérage des paramètres significatifs. Cela suppose donc qu'un nombre suffisant de paramètres soient accessibles, laissant à l'utilisateur la liberté de choisir une stratégie expérimentale, de faire des erreurs, de réfléchir, ...

  Dans quelle mesure doit-on simplifier ou compliquer le modèle qui est mis en oeuvre ? D'un côté on peut choisir de simplifier la représentation de la réalité, pour que l'élève puisse se concentrer sur l'essentiel. Mais cela signifie aussi que la situation analysée n'est peut-être plus qu'une image grossière de la réalité, sans valeur pédagogique véritable. D'un autre côté, on peut choisir un modèle plus complexe qui fournit une image plus fidèle de la réalité. Mais cela entraîne souvent que le programme devient difficile à

utiliser et/ou s'exécute lentement. De plus, l'élève se trouvant confronté à un nombre trop élevé de décisions à prendre, s'embarque dans des opérations inutiles et finit par ne plus comprendre l'objet même du travail.
Dans ce contexte, l'utilisation de l'ordinateur présente un grand avantage, c'est qu'on peut intégrer dans le logiciel la possibilité d'adapter le degré de réalisme de la simulation.

- La structure du programme ne doit pas être trop linéaire, de sorte que l'utilisateur puisse cheminer à son gré dans la simulation. Grâce à l'ordinateur, on peut se permettre de faire aborder un problème aux élèves par la méthode d'essais et erreurs, les laisser recommencer de nombreuses fois la même chose, leur apprendre à tester un seul paramètre à la fois, ...
- Si l'on veut que l'élève puisse vraiment cheminer à son propre rythme, progressant à coups d'essais et d'erreurs, il faut lui donner le temps d'explorer un peu. Il est donc intéressant que le programme de simulation soit conçu de sorte que l'on puisse l'utiliser de manière interrompue, par étapes. Ceci implique que l'on puisse sauvegarder l'état du système, à la fin d'une séance, pour reprendre le travail la fois suivante, à l'endroit exact où on l'avait interrompu.
- Le logiciel peut contenir une aide en ligne suffisamment fournie. Ainsi l'élève ne monopolise pas l'enseignant pour lui poser des questions auxquelles le programme peut aussi bien fournir la réponse.
- Dans beaucoup de cas, les simulateurs profitent des nombreuses possibilités de l'ordinateur pour que l'élève puisse réaliser/visualiser des choses qui sont impossibles (ou difficiles) à réaliser/visualiser dans le monde réel : faire avancer le temps plus vite, retourner en arrière pour annuler une opération, ...

1.3 Simulateur d'un banc d'optique

1.3.1 Objectif "*Simulation*"

Dans le cadre de ce mémoire, nous nous sommes intéressés à la réalisation d'un simulateur de banc d'optique.
Il est évident que l'utilisation d'un simulateur d'expériences scientifiques ne doit pas remplacer l'expérimentation concrète, si celle-ci est réalisable : l'utilisation d'instruments et de techniques, la prise de conscience de la complexité qu'apportent les manipulations expérimentales réelles, sont irremplaçables. Mais il faut bien se rendre compte que le matériel optique est très cher et parfois délicat à utiliser, ce qui implique que toutes les écoles n'ont pas la possibilité d'acheter le matériel expérimental nécessaire, mais ils peuvent tout de même se rabattre sur des logiciels de simulation. Et même dans les écoles qui ont des laboratoires bien équipés à leur disposition, le simulateur trouve toujours sa place en tant que complément des travaux pratiques. Ceci pour la simple raison que l'utilisation de l'ordinateur permet de proposer à l'élève une multitude d'actions et de possibilités qui ne sont pas disponibles sur un vrai banc d'optique. Nous reviendrons sur ce point essentiel un peu plus loin dans le texte.

Le premier objectif de notre logiciel est de permettre à des élèves du secondaire de réaliser des montages optiques sur un banc d'optique et d'observer ainsi les phénomènes physiques qui ont été décrits au cours théorique. Dans le mini-monde que représente notre logiciel, nous avons envisagés la réalisation de montages composés d'un objet optique, d'une ou plusieurs lentilles et d'un écran. Nous nous sommes limités aux lentilles minces pour deux raisons :

- Les élèves du secondaire étudient la plupart du temps les phénomènes de réflexion (miroirs plans et sphériques) et de réfraction (surfaces planes, dioptres sphériques et lentilles sphériques minces). De tous ces sujets il nous a semblé que celui des lentilles minces étaient le plus intéressant, surtout du point de vue des montages que l'on peut réaliser sur un banc d'optique : système de plusieurs lentilles placées l'une derrière l'autre, construction d'instruments optiques (microscope, appareil de projection, ...), visualisation des aberrations, ...
- On ne voulait pas que notre logiciel soit uniquement un programme de simulation, mais qu'on puisse également faire le lien entre les phénomènes observés expérimentalement et les équations régissant ces phénomènes (voir plus loin). Nous avons donc consacré une partie de notre temps à la réalisation de ce deuxième aspect plutôt que de pousser plus à fond l'aspect

simulation.

L'aspect simulation de notre logiciel permet donc à l'élève de placer, comme il le souhaite, les différents composants de son montage sur le banc, et de réaliser ensuite toute une série d'actions qu'il pourrait également réaliser sur un vrai banc : déplacer les objets, rechercher l'image à l'aide de l'écran qu'il déplace sur le banc, supprimer/ajouter une ou plusieurs lentilles, ... Mais à côté de cela, l'élève va également pouvoir faire des choses qui ne sont pas possibles (ou difficilement réalisables) sur un vrai banc. C'est ici que l'on remarque un des grands avantages du simulateur par rapport à un banc d'optique réel : on ajoute des fonctionnalités qui augmente l'intérêt pédagogique du logiciel :

- On a déjà vu que l'élève peut trouver l'emplacement de l'image à l'aide de l'écran, mais l'élève peut également demander à l'ordinateur de dessiner l'image et de l'afficher de façon permanente. Ainsi, à chaque fois que l'élève change un paramètre du montage il ne doit pas "perdre son temps" à déplacer son écran pour trouver la nouvelle image, mais il observe directement les conséquences de sa modification.
- Grâce à cet affichage par l'ordinateur, l'élève peut même visualiser des images virtuelles que sur un vrai banc il ne saurait jamais voir, car il ne saurait pas les recueillir sur son écran.
- Dans un laboratoire, l'élève a un nombre limité de lentilles à sa disposition. Le simulateur dispose de toutes les lentilles possibles vu que l'élève peut "construire" ses propres lentilles en introduisant les paramètres caractéristiques de celles-ci. Ainsi l'élève peut faire varier un seul paramètre à la fois et étudier son influence sur le résultat final.
- Lorsque l'élève construit un montage à une ou plusieurs lentilles, il peut observer comment l'image finale est obtenue en demandant l'affichage des images intermédiaires qui sont formées à travers chacune des lentilles, ainsi de que quelques uns des rayons lumineux qui les forment.
- L'élève peut changer dans une certaine mesure le niveau de réalisme de la simulation : l'élève décide si le logiciel tient ou ne tient pas compte des défauts optiques des lentilles et de leurs conséquences. Ceci lui permet de bien voir les différences qui existent entre un montage réalisé avec des lentilles parfaites et ceux réalises avec des lentilles réelles (et donc imparfaites). Sur un vrai banc d'optique on n'a pas ce choix : on travaille avec les lentilles qu'on a à disposition et qui présentent la plupart du temps des défauts optiques.
- L'élève a la possibilité de sauvegarder ou d'imprimer son montage pour

que la fois suivante il ne doive pas tout recommencer ou pour simplement conserver une trace de ce qu'il a fait.

1.3.2 Objectif "*Plus que de la simulation...*"

Jusqu'à présent, nous avons décrit notre premier objectif qui était l'aspect simulation de notre logiciel. Nous avons déjà remarqué que cet aspect présente pas mal d'avantages au niveau pédagogique, mais il nous a semblé intéressant de réaliser un logiciel qui aille plus loin que la simple simulation et ceci pour augmenter son intérêt pédagogique et également son originalité.

Le deuxième objectif de notre logiciel est d'y intégrer, parallèlement au simulateur proprement dit, les équations qui régissent les phénomènes optiques, pour que l'élève se rende mieux compte du lien qui existe entre la théorie et la pratique. A l'aide de la simulation l'élève va pouvoir observer ce qui se passe lorsqu'il modifie un paramètre de son montage, et grâce à la partie relative aux équations il va pouvoir comprendre pourquoi le résultat final est bien celui qu'il observe. Pour ce faire, un montage va pouvoir être entièrement représenté à l'aide des valeurs numériques des paramètres caractéristiques des différents composants (objet optique, lentille(s), ...). On n'est pas obligé de disposer de toutes les valeurs de tous les paramètres, mais bien d' un nombre suffisant pour que le système soit déterminé. Ensuite, on peut jongler avec ces valeurs et avec les équations : calcul des paramètres qu'on ne connaît pas, introduction de nouvelles valeurs et calcul des changements que cela entraîne, ...

Le fait de réunir ces deux aspects au sein d'un même logiciel augmente naturellement considérablement les possibilités d'utilisation et donc les possibilités d'investigation de l'élève. Il peut par exemple se poser un problème ("J'aimerais bien faire un montage qui fasse *cela*."), trouver à l'aide des équations comment il peut obtenir ce résultat et ensuite construire le montage pour vérifier si c'est bien exact. Ou bien, une fois qu'il a réalisé un montage, il peut se poser la question de savoir ce qu'il doit changer pour obtenir un autre résultat, par exemple une image deux fois plus grande. D'un côté il peut procéder expérimentalement par la méthode d'essais et d'erreurs en jouant avec le simulateur. D'un autre côté, il peut utiliser les équations pour résoudre son problème.
On voit directement que l'intégration de cette partie "équations" apporte énormément à l'aspect pédagogique du logiciel : l'élève dispose d'une grande liberté pour décider lui-même de sa démarche, chercher un fil conducteur, mesurer, faire des essais et des erreurs, ... Ce sont toutes des composantes essentielles de tout

processus d'apprentissage.
Ce travail d'investigation effectué par l'élève peut également être stimulé en lui proposant des exercices à résoudre. C'est pourquoi nous proposons dans notre logiciel deux modes de fonctionnement : le mode "Libre" qui laisse toute liberté d'action à l'élève et un autre, le mode "Problème", qui est plus directif et qui permet également un suivi des élèves par l'enseignant.

Mode "Libre"

Dans ce mode, l'élève utilise librement les parties "simulation" et "équations" du logiciel. Comme on l'a déjà vu plus haut, les possibilités qui s'offrent ainsi à l'élève sont très nombreuses : résoudre des problèmes qu'il s'est posés, comprendre l'influence de chaque paramètre sur le résultat final, ... L'initiative doit toujours venir de l'élève lui-même, afin qu'il utilise le simulateur comme un vrai terrain d'expérimentation.
Pour arriver à cela, l'élève peut demander différentes actions à l'ordinateur :

- Lorsque l'élève a introduit un certain nombre de valeurs numériques dans la partie "équations", l'ordinateur peut vérifier la cohérence de toutes ces données, c'est-à-dire contrôler si les données vérifient bien les différentes équations.
- L'ordinateur peut effectuer la résolution d'une équation pour laquelle l'élève a introduit tous les paramètres dont il dispose. La résolution réalisée par l'ordinateur se limite à la résolution d'une équation à une inconnue, ce qui est suffisant dans la plupart des cas.
- L'élève peut demander au logiciel de transférer les données de son montage vers la partie "équations", ainsi l'élève pourra disposer des valeurs numériques qui représentent son montage.
- De même, il peut demander le transfert inverse, c'est-à-dire la réalisation du montage correspondant aux données numériques introduites dans la partie "équations".

Concernant ces deux transfert, on a été confronté à un choix : ces transfert se font-ils automatiquement par l'ordinateur (à la demande de l'élève) ou manuellement par l'élève ? Un transfert manuel signifierait que l'élève devrait à chaque fois redéfinir ou bien tout son montage expérimental ou bien toutes les données numériques. Dans certains cas, cela peut représenter un travail assez fastidieux dans lequel une erreur de frappe est vite arrivée. C'est pour cela et aussi pour que l'élève puisse plus se concentrer sur le résultat du transfert que nous avons opté pour la solution automatique. Illustrons maintenant à l'aide d'un exemple comment l'élève pourrait utiliser ces transferts de données :

L'élève réalise un montage quelconque et ensuite il se demande comment il devrait modifier les paramètres de la lentille pour obtenir une image qui soit deux fois plus grande. Pour ce faire, il commence par transférer son montage dans la partie "équations", ce qui lui permet de faire les calculs nécessaires : il choisit le(s) paramètre(s) à changer, il résout les équations, ... Finalement, pour vérifier s'il obtient bien le résultat escompté, il fait un transfert des nouvelles valeurs numériques vers le banc.

Mode "Problème"

Dans le mode "Problème", qui est le mode plus directif, on présente à l'élève un exercice qu'il doit résoudre. Chaque exercice consiste à déterminer un certain nombre de valeurs numériques qui sont inconnues et à réaliser le montage optique qui correspond à l'énoncé. Pour obtenir ces résultats, l'élève va devoir utiliser aussi bien la partie "simulation" que la partie "équations".
En ce qui concerne les valeurs numériques, il y en a certaines qu'on ne trouve qu'à l'aide de la partie "équations", notamment en résolvant une ou plusieurs de ces équations. Les autres valeurs peuvent soit être calculées de la même façon à l'aide des équations, soit être trouvées expérimentalement en utilisant la simulation du banc. C'est là un choix qui est laissé à l'élève.
Une fois que l'élève a terminé la résolution, il demande au logiciel d'en faire la correction. L'aspect *correction* d'un exercice est du point de vue pédagogique une des partie les plus complexes dans un logiciel éducatif. Avec un ordinateur il n'est pas difficile de contrôler si la réponse donnée par l'utilisateur est correcte, mais ce qui est bien plus compliqué, c'est de vérifier le raisonnement utilisé.
Comment avons-nous abordé ce problème ? La correction d'un exercice dont la réponse proposée n'est pas la bonne, se déroule en trois étapes :

- Dans un premier temps, le logiciel signale à l'élève que sa réponse n'est pas juste. A ce moment-là, deux possibilités sont offertes à l'élève : ou bien il essaye de trouver par soi-même son erreur, ou bien il passe à l'étape suivante.
- Pour essayer de guider l'élève dans sa correction, le programme lui précise quelles parties de sa réponse ne sont pas correctes. A nouveau, il y a les deux mêmes possibilités qui s'offrent à l'élève : chercher tout seul ou passer à l'étape suivante.
- Finalement, la solution correcte est présentée à l'élève.

Ce procédé est naturellement assez simple puisqu'on n'essaye jamais de localiser l'endroit d'où provient la faute. Cette étape-là doit être effectuée par l'étudiant,

soit tout seul, soit avec l'aide de l'enseignant.

Ce mode "Problème" nous permet aussi d'intégrer dans le logiciel un suivi des élèves par l'enseignant, c'est-à-dire que l'enseignant peut consulter pour chacun de ses élèves quels exercices ils ont résolus et après combien d'essais ils ont obtenu la bonne réponse. Pour ce faire, il faut que les élèves se fassent connaître lorsqu'ils démarrent le logiciel et que toutes les données concernant la résolution des exercices (numéro de l'exercice, assistance de l'ordinateur lors de la correction, ...) soient enregistrées dans une base de données.
L'enseignant, qui doit aussi se faire connaître auprès du logiciel, peut donc consulter cette base de données pour faire une évaluation de ces élèves. Il y a encore une autre chose que l'enseignant peut faire et que les élèves ne peuvent pas faire : c'est ajouter de nouveaux exercices dans la base de données (énoncés avec réponses correctes correspondantes).

1.3.3 Projet réellement concrétisé

Ce que nous avons décrit jusqu'à présent est le projet tel que nous l'avions vu au début. Malheureusement, au niveau de la réalisation nous n'avons pas pu aller aussi loin que prévu. Certaines parties n'ont donc pas été implémentées :

- Dans les montages que les élèves réalisent, on ne tient pas compte des défauts optiques des lentilles, ce qui veut dire que les élèves ne peuvent pas visualiser les différents types d'aberrations.
- Il n'y a pas de bases de données dans laquelle on enregistre les données concernant la résolution des exercices.
- L'enseignant n'a pas la possibilité d'ajouter de nouveaux exercices au logiciel.

Chapitre 2

Introduction à l'optique géométrique

2.1 Introduction

Ce chapitre n'a pas pour but d'exposer tous les principes de l'optique géométrique, mais bien d'aborder les concepts qui seront utilisés lors de la réalisation du simulateur [JW76]. Les hypothèses faites au niveau de l'optique dans ce chapitre sont des hypothèses qui seront également conservées lors de la mise en application dans le logiciel.

On commencera tout d'abord par expliquer ce qu'on entend par "optique géométrique". Ensuite on parlera du phénomène de réfraction (sur une surface sphérique), pour terminer par les lentilles sphériques.

2.2 Théories optiques

Au cours de l'histoire de la physique différentes théories optiques ont été émises, dont les trois plus importantes sont les suivantes :

2.2.1 Optique géométrique

Cette théorie conçoit la lumière comme étant composé de petites particules et elle admet que les rayons lumineux ont les caractéristiques suivantes :

- ils sont indépendants les uns des autres ;
- ils sont des lignes droites dans les milieux homogènes ;
- les changements de direction dus à la réflexion et à la réfraction se font suivant des lois bien déterminées.

Cette manière de concevoir l'optique n'est qu'une représentation simpliste de la réalité, mais qui nous permet d'expliquer un grand nombre de faits physiques (entre autres ceux que nous allons simuler). Cette théorie fut spécialement soutenue par Newton (1642–1727).

2.2.2 Optique ondulatoire

L'optique ondulatoire, initiée par Fresnel (1788–1827), conçoit une source de lumière comme un centre qui produit des ondes qui se propagent. Elle a permis d'expliquer certains phénomènes jusqu'à alors inexpliqués, comme par exemple le phénomène de diffraction par une fente mince qui donne lieu à des franges d'interférence.

2.2.3 Optique des photons

Depuis le début du 20ème siècle, à la suite de nouvelles expériences, une troisième théorie, l'optique des photons a fait son apparition. Elle conçoit la lumière comme un flux de quanta émis par la source : les photons. Cette théorie montre qu'une interprétation complète de certains phénomènes ne peut être obtenue qu'en conservant à la fois l'aspect ondulatoire et l'aspect corpusculaire de la lumière (l'onde permettant de calculer la probabilité pour qu'un corpuscule se manifeste).

2.3 Réfraction par un dioptre sphérique

2.3.1 Phénomène de réfraction

On appelle "réfraction" le changement de direction que subit un faisceau de lumière en passant d'un milieu homogène dans un autre. Lorsque l'on étudie la réfraction de la lumière par une surface plane (voir figure 2.1), on aboutit aux conclusions suivantes :

1. Le rayon réfracté se trouve dans le plan déterminé par le rayon incident et la normale à la surface plane. Le rayon réfracté et le rayon incident sont situés de part et d'autre de la normale.
2. Le rapport $\sin i / \sin r$ (du sinus de l'angle d'incidence au sinus de l'angle de réfraction) est constant pour une même radiation monochromatique et pour deux milieux donnés. Il est habituel de représenter cette constante par $n_{1,2}$ qui est appelée indice de réfraction relatif du second milieu par rapport au premier, pour la lumière monochromatique considérée. Cet indice relatif étant le rapport des indices absolus des deux milieux, on peut réécrire le rapport de sinus comme suit :

$$\frac{\sin i}{\sin r} = \frac{n_2}{n_1}$$

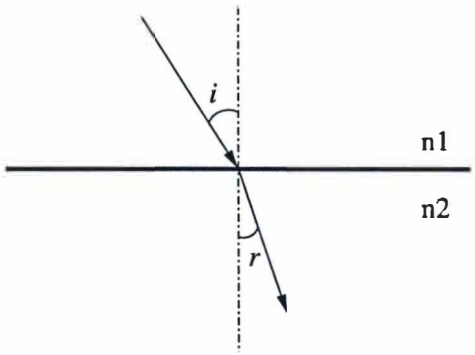

FIG. 2.1: Réfraction d'un rayon lumineux sur une surface plane

2.3.2 Réfraction par un dioptre sphérique

Un dioptre sphérique est la surface qui sépare deux milieux d'indices de réfraction absolus différents (respectivement n_1 et n_2) (voir figure 2.2). Le *centre de courbure* C du dioptre est le centre de la sphère et le *sommet* O est le pôle

de la calotte sphérique. La droite passant par O et C est appelée *axe optique*. Le rayon de courbure R du dioptre est défini comme la distance entre le sommet et le centre de courbure de la lentille. Nous prenons comme convention que les rayons lumineux se propagent de la gauche vers la droite. En se basant sur cette convention, on a que :

- si le centre de courbure se trouve à droite du sommet, le rayon de courbure sera positif;
- si le centre de courbure se trouve à gauche du sommet, le rayon de courbure sera négatif.

Ce qui nous intéresse lorsqu'on étudie la réfraction de la lumière, c'est de voir ce qui se passe avec la lumière qui est émise par un objet situé à une certaine distance p du dioptre. Un objet pouvant être considéré comme étant formé par un ensemble de points lumineux, nous allons faciliter notre raisonnement en prenant comme objet un simple point.
Ce point est dit *objet* par rapport au dioptre : il est le point d'intersection des rayons incidents. Les rayons lumineux vont être réfractés par le dioptre et l'expérience nous montre que les rayons réfractés vont ou bien converger (i.e. se diriger vers l'axe optique) ou bien diverger (i.e. s'éloigner de l'axe optique). Ce sont ces rayons réfractés qui vont former l'*image* du point : elle sera formée au point d'intersection des rayons réfractés.

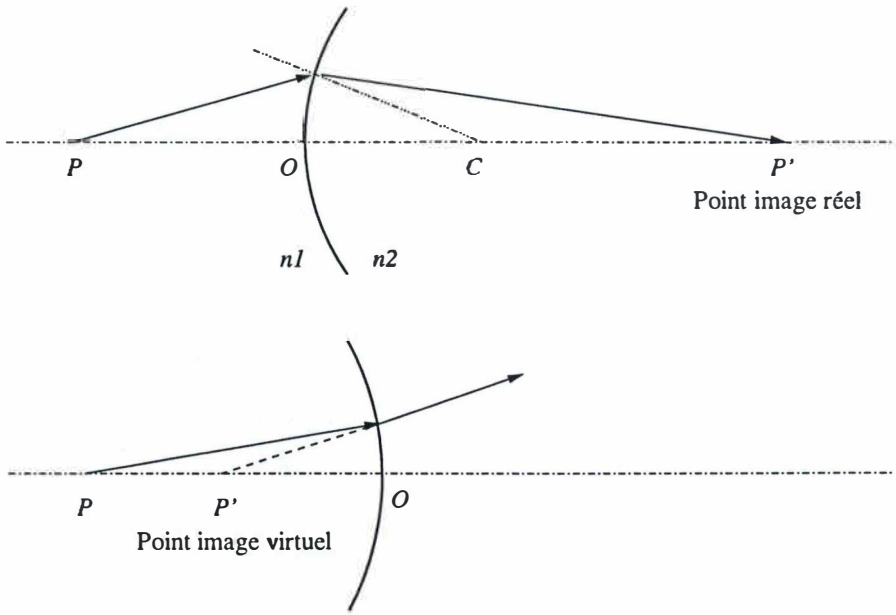

FIG. 2.2: Réfraction sur un dioptre sphérique

Un point image peut être de deux natures différentes (voir figure 2.2). Il est dit *réel* si les rayons réfractés passent réellement par ce point d'intersection. Et un point image est dit *virtuel* s'il est le point de concours du prolongement des rayons réfractés : les rayons s'y croiseraient s'ils étaient prolongés dans le sens opposé à leur direction de propagation. Ceci a naturellement comme conséquence qu'une image réelle peut être recueillie sur un écran, tandis qu'une image virtuelle ne le peut pas.

La réfraction à la séparation des milieux se manifeste en chaque point comme sur un plan tangent, ce qui nous permet d'utiliser les lois de la réfraction sur une surface plane. En prenant comme hypothèse que les rayons lumineux ne sont pas trop inclinés (par rapport à l'axe optique)[1], on peut montrer que la relation reliant la distance p (distance entre le point objet et le dioptre) à la distance p' (distance entre le dioptre et le point image p') s'écrit de la façon suivante :

$$\frac{n_1}{p} + \frac{n_2}{p'} = (n_2 - n_1)\frac{1}{R}$$

Lorsque la valeur de p' est positive, cela veut dire que l'image est formée à droite du dioptre et qu'elle est également réelle. Dans le cas où p' est négatif, l'image sera située à gauche du dioptre.

A un dioptre on peut encore associer deux grandeurs qui sont définies comme suit :

- le *foyer objet* F (ou *premier point focal*) est la position d'un point objet sur l'axe optique telle que les rayons réfractés soient parallèles à l'axe optique, ce qui revient à former l'image du point à l'infini ($p' = \infty$). On a aussi la distance focale f entre le foyer objet et le sommet du dioptre :

$$p' = \infty \qquad f = \frac{n_1}{(n_2 - n_1)}R$$

- si les rayons incidents sont parallèles à l'axe optique, ce qui revient à avoir un objet très éloigné de la surface sphérique ($p = \infty$), les rayons réfractés passent par un point F' appelé *foyer image* (ou *second point focal*). On trouve la distance focale f' entre le sommet du dioptre et le foyer image :

$$p = \infty \qquad f' = \frac{n_2}{(n_2 - n_1)}R$$

[1]Dans ce cas on peut faire une approximation du sinus d'un angle par la valeur de l'angle : $\sin\theta \approx \theta$, où la valeur de θ est exprimée en radians.

Dans ce qui précède on a toujours parlé d'un "point" objet, c'est-à-dire une source lumineuse ponctuelle, mais on peut évidemment étendre cela à un objet lumineux plus grand en le considérant comme étant un ensemble de points lumineux auxquels on applique chaque fois les lois de la réfraction. De même son image sera formée par l'ensemble des images de ses points.

2.4 Lentilles sphériques

2.4.1 Généralités

Une lentille sphérique est un milieu transparent, limité par des faces sphériques, bien que l'une des faces puisse être plane (ce qui équivaut à un rayon de courbure infini). Elle est donc la juxtaposition de deux dioptres sphériques : un rayon lumineux incident subit deux réfractions lors de sa traversée.
On distingue deux catégories de lentilles sphériques : les lentilles convergentes, que l'on représente dans les dessins par un trait terminé par deux flèches dont la pointe est dirigée vers l'extérieur, et les lentilles divergentes, que l'on représente par un trait terminé par deux flèches dont la pointe est dirigée vers l'intérieur. Pour les lentilles divergentes, on a que les rayons réfractés vont s'éloigner de l'axe optique, tandis que pour les lentilles convergentes, ils vont se diriger vers l'axe optique.

Pour une lentille donnée, on peut définir les grandeurs suivantes :

- *centres de courbure* (C_1 et C_2) et *rayons de courbure* (R_1 et R_2) : l'indice 1 indique que cela se rapporte à la surface de la lentille qui est traversée en premier par les rayons lumineux (surface de gauche dans notre convention) et l'indice 2 à la deuxième surface (surface de droite) ;
- L'*axe principal* qui est la droite qui joint ses centres de courbure ;
- L'*ouverture* de la lentille : le rapport du diamètre de la lentille à sa distance focale.
- La *puissance* Π de la lentille est, par définition, l'inverse de sa distance focale. On a que :

$$\Pi = \frac{1}{f}$$

  Si on mesure f en mètre, alors la puissance est exprimée en dioptrie.

2.4.2 Formules pour les lentilles minces

Commençons par faire les trois hypothèses suivantes qui vont nous permettre de simplifier le problème pour des raisons pédagogiques :

- Admettons que les milieux de part et d'autre de la lentille sont identiques et leur indice de réfraction égal à un (comme c'est le cas pour l'air) tandis que l'indice de réfraction de la lentille est n. Cette première hypothèse est souvent vérifiée vu que c'est la situation dans laquelle on se retrouve (la plupart du temps) lorsqu'on place une lentille sur un banc d'optique.

- On ne considère également que les lentilles minces, c'est-à-dire dont l'épaisseur est très petite devant les rayons de courbure. Cette hypothèse-ci est beaucoup plus contraignante, mais nous permet de simplifier fortement les calculs. Sans cette simplification, chaque fois que l'on considère des distances par rapport à une lentille, on devrait les mesurer à partir des sommets O_1 et O_2 suivant les cas. Mais comme les lentilles considérées sont minces, on peut négliger la distance d qui sépare les deux sommets, ce qui revient à mesurer toutes les distances à partir d'une origine commune O (*centre optique de la lentille*).
- Les lentilles sont parfaites : on ne tient pas compte des défauts possibles d'une lentille (voir paragraphe 2.4.5).

En se basant sur les résultats obtenus pour les dioptres sphériques et en tenant compte des hypothèses que l'on vient de faire, on trouve que les deux distances focales f et f' sont égales :

$$\frac{1}{f} = \frac{1}{f'} = (n_{1,2} - 1)(\frac{1}{R_1} - \frac{1}{R_2})$$

et aussi que :

$$\frac{1}{f} = \frac{1}{p} + \frac{1}{p'}$$

Pour une lentille convergente, on obtient toujours des distances focales f et f' positives : le foyer objet F se trouve donc à gauche de la lentille et le foyer image F' à droite. Dans les cas d'une lentille divergente, les distances focales sont négatives : les foyers objet et image sont alors respectivement à droite et à gauche de la lentille.

2.4.3 Position, grandeur et sens de l'image

Considérons comme objet une droite qui est perpendiculaire à l'axe principal d'une lentille convergente. Nous allons maintenant voir qu'il est possible, à partir d'une construction géométrique très simple, de déterminer l'image de cette droite. C'est cette construction géométrique qui sera utilisée dans le logiciel pour déterminer la position de l'image.
On a vu que l'image d'un point est le point de convergence de tous les rayons réfractés. Pour trouver l'image, il suffit donc de trouver le point de rencontre de deux d'entre eux. Pour que la construction géométrique soit la plus simple possible, on choisit les deux rayons incidents suivants :

- le rayon passant par le centre optique : ce rayon subit la réfraction comme s'il traversait un dioptre plan d'épaisseur nulle. Le rayon n'étant pas dévié, le rayon réfracté est dans le prolongement du rayon incident.
- le rayon parallèle à l'axe principal : ce rayon vient d'un point situé à l'infini et le rayon réfracté qui lui correspond passe par le foyer F'.

En prenant l'intersection de ces deux rayons réfractés, on trouve la position de l'image (voir figure 2.3).

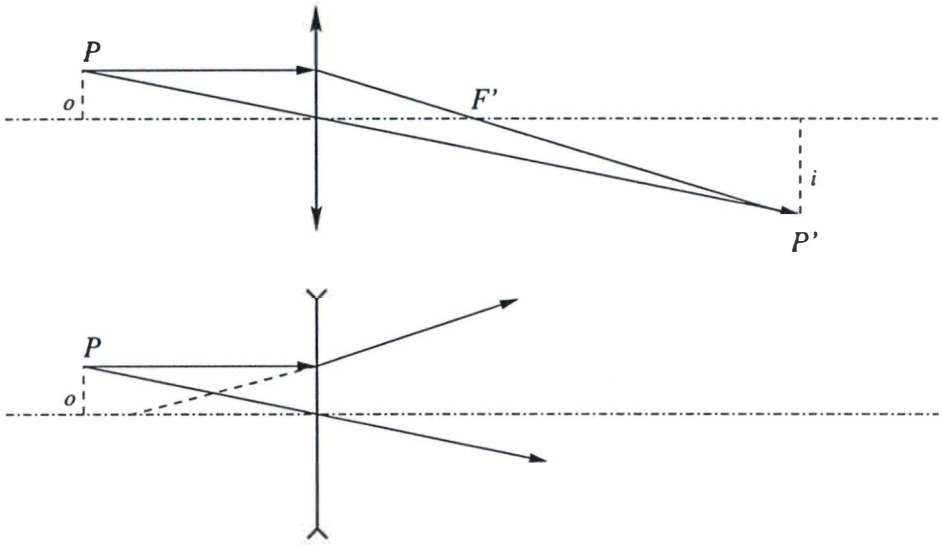

FIG. 2.3: Construction géométrique déterminant la position de l'image

La taille de l'objet/l'image se mesure toujours par rapport à l'axe optique : si il/elle se trouve au-dessus de l'axe optique, il/elle aura une taille positive. Dans le cas contraire, il/elle aura une taille négative. Le rapport entre la taille de l'objet, notée o, et celle de l'image, notée i, nous donne l'agrandissement de l'objet :

$$M = \frac{i}{o} = \frac{-p'}{p}$$

L'agrandissement nous indique également le sens de l'image : si M est négatif, cela veut dire que l'image est renversée par rapport à l'objet et si M est positif, alors l'objet et l'image se trouvent dans le même sens.
Finalement, en faisant une étude géométrique on peut construire le tableau suivant[2] qui est valable pour une lentille convergente :

[2]Dans le cas de la dernière ligne du tableau, on a affaire à un objet de type virtuel : il est le point d'intersection des rayons incidents prolongés au-delà de la lentille.

OBJET	***IMAGE***			
Position	*Position*	*Sens*	*Nature*	*Grandeur*
à $-\infty$	en F'	renversée	réelle	nulle
de $-\infty$ à $2F$	de F' à $2F'$	renversée	réelle	$i < o$ croissante
en $2F$	en $2F'$	renversée	réelle	$i = o$
de $2F$ à F	de $2F'$ vers $+\infty$	renversée	réelle	$i > o$ croissante
en F	à ∞ (+ ou -)	renversée ou droite	réelle ou virtuelle	infinie
de F à O	de $-\infty$ à O	droite	virtuelle	$i > o$ décroissante
de O à $+\infty$	de O à F'	droite	réelle	$i < o$ décroissante

2.4.4 Système de lentilles

On appelle système de lentilles, plusieurs lentilles qui ont le même axe principal.
Pour trouver l'image finale qui est formée par cet ensemble de lentilles, on procède de la façon suivante :

- On considère l'objet optique et la première lentille (celle qui se trouve le plus à gauche) et on fait comme si toutes les autres lentilles n'étaient pas présentes sur le banc.
- On calcule l'image qui serait formée par cette lentille.
- Ensuite on recommence le même processus avec la lentille suivante, mais en prenant comme objet optique l'image que l'on vient de trouver.
- Une fois qu'on est passé par toutes les lentilles, la dernière image que l'on a obtenue est celle de l'objet optique à travers toutes les lentilles.

Comme on l'a déjà signalé dans le paragraphe précédent, il peut arriver que l'on soit confronté à une objet qui se trouve à droite de la lentille et qui est donc un objet virtuel.

2.4.5 Défauts des lentilles

Dans nos hypothèses nous avons supposé que les lentilles étaient optiquement parfaites, ce qui est loin d'être le cas dans la réalité. Les défauts des lentilles donnent lieu à un certain nombre d'*aberrations* :

- *Aberration de sphéricité* : les rayons d'un faisceau parallèle qui sont réfractés sur les bords de la lentille (rayons marginaux) ne convergent pas au même point que les rayons de ce même faisceau réfractés au centre (rayons

centraux). Le point de convergence des rayons marginaux est plus rapproché de la lentille que celui des rayons centraux. L'aberration de sphéricité est d'autant plus accentuée que l'ouverture de la lentille est grande. Pour limiter ce problème on utilise ou bien des lentilles de petite ouverture, ou bien des lentilles de plus grande ouverture, mais devant lesquelles on place un diaphragme (c'est-à-dire une ouverture de diamètre réglable).

- *Aberration d'obliquité* : ce défaut se manifeste avec des faisceaux obliques et étroits. Il consiste en ceci que les rayons réfractés correspondant à un faisceau incident oblique et étroit, ne convergent pas en un point, mais en deux petites droites, appelées *focales* et de directions perpendiculaires.
- *Aberration chromatique* : la direction du rayon réfracté dépende de la couleur du rayon incident. Cette aberration résulte du fait que l'indice de réfraction n'est pas le même pour toutes les longueurs d'onde : il augmente régulièrement au fur et à mesure que la longueur d'onde de la lumière diminue. Lorsqu'on utilise un objet émettant une lumière comportant plusieurs longueurs d'onde (la lumière blanche par exemple), on aura que les bords de l'image formée par la lentille paraissent colorés. Pour éliminer cette aberration, on peut filtrer la lumière émise par la source pour ne laisser passer qu'un longueur d'onde déterminée, ou bien utiliser directement une source monochromatique.

2.4.6 Banc d'optique

Un banc d'optique, comme celui qu'on simule dans notre logiciel, se présente comme un long rail rectiligne et (la plupart du temps) gradué, que l'on place dans une chambre noire. Les différents objets[3] (lentilles, écran, lampe, ...) sont chaque fois fixés sur un pied de sorte qu'on puisse les déposer facilement sur ce rail.

Comment réalise-t-on un montage optique sur un tel banc ? Tout d'abord on commence par placer une lampe à l'extrême gauche, dont la lumière est dirigée vers la droite. Ensuite on place une diapositive à droite de la lampe : cette diapositive est en réalité une plaque avec au milieu une fente circulaire, une fente rectiligne ou autre. La diapositive éclairée par la lampe fait office d'objet optique. Ensuite on place les différentes lentilles, en veillant bien à aligner tous les sommets. Et finalement à l'extrême droite on place l'écran.

[3]Dans la suite du document, nous utiliserons le terme "objet optique" lorsqu'on veut désigner l'objet optique lumineux qui donne lieu à une image et le terme "objet" lorsqu'on parle de toutes les composantes que l'on peut placer sur le banc : l'objet optique, les lentilles et l'écran.

Chapitre 3

Analyse fonctionnelle

3.1 Introduction

Dans ce chapitre nous allons aborder l'analyse fonctionnelle de notre logiciel. Puisque dans un simulateur la partie interface représente une partie essentielle, nous avons découpé notre analyse en deux parties : l'analyse de l'interface (Comment va-t-on présenter les différents éléments à l'utilisateur ?) et l'analyse fonctionnelle proprement dite (Quelles sont les fonctionnalités du logiciel ?).
Quand on regarde le logiciel que nous voulons réaliser, on se rend compte qu'il est composé de trois grandes parties : la simulation du banc d'optique, la partie relative aux équations et la partie relative aux exercices. C'est selon cette découpe que nous allons faire nos deux analyses. Remarquons cependant que pour l'analyse fonctionnelle on rajoutera une toute petite partie où on parlera des fonctionnalités qui concernent les trois parties du logiciel en même temps.

3.2 Analyse de l'interface

3.2.1 Introduction

Ici aussi, on va se baser sur la découpe en trois parties que l'on a faite précédemment. Au niveau de l'interface, cela va impliquer que chaque partie du logiciel se verra attribuer une fenêtre séparée.

3.2.2 Banc d'optique

Il est évident que pour cette partie on aura une interface essentiellement graphique, puisque pour que l'utilisation de la simulation soit la plus naturelle possible, il faut que l'utilisateur puisse manipuler directement la représentation

graphique des différents objets. Ceci se fera la plupart du temps à l'aide de la souris, parfois également à l'aide du clavier, et dans certains cas on aura les deux possibilités.

Comment va-t-on représenter notre banc d'optique ?

Tout d'aborder on doit décider de la vue que l'élève aura du banc. Nous avons décidé de représenter le banc de sorte que le regard de l'élève soit dirigé perpendiculairement par rapport à l'axe optique du système, qui lui se trouve dans une position horizontale. Ainsi on obtient le même aspect que celui qui avait été adopté dans les figures du chapitre 2.

Dans le chapitre 2, nous avons vu que lorsqu'on réalise un montage avec plusieurs lentilles, on doit bien veiller à centrer les lentilles sur le même axe optique. Il est donc intéressant d'afficher une droite qui servira d'axe optique pour montrer à l'élève que les sommets de toutes les lentilles se trouvent sur cet axe.

Quand on travaille sur un banc d'optique les notions de position et de distance sont très importantes. C'est pourquoi l'élève dispose d'une règle graduée affichée au bas de la fenêtre. Naturellement cette règle n'est pas toujours suffisante lorsqu'il veut connaître une position avec plus de précision. Ce problème est résolu en affichant continuellement les coordonnées de la position actuelle du curseur.

Vu que les différentes distances (distance focale, distance inter-lentille, ...) peuvent fortement varier d'un montage à l'autre, l'élève dispose d'une fonction "zoom" de sorte qu'il puisse visualiser l'entièreté de son montage. Ce zoom est uniquement un zoom horizontal, ce qui nous permet de garder un objet de taille observable. Cette restriction ne pose pas de problèmes dans l'état actuel du logiciel où on ne tient pas compte des aberrations. Effectivement, si on voulait pouvoir visualiser les aberrations, la fonction de zoom devrait être aussi bien horizontale que verticale, car sinon on obtiendrait une vue biaisée de la réalité.
Ce zoom pourrait être déclenché automatiquement par l'ordinateur, ou bien explicitement par l'élève. D'un point de vue pédagogique, il semble plus intéressant de retenir la deuxième solution. Envisageons le cas d'un montage où l'objet optique et les différentes lentilles peuvent être affichés ensemble dans la fenêtre, mais où l'image se formerait en dehors de la portée de la fenêtre. Dans le cas du zoom automatique, l'ordinateur s'en rend compte, et va immédiatement adapter l'échelle pour que l'élève puisse tout voir sur l'écran de l'ordinateur. Dans le cas d'un zoom manuel, c'est l'élève qui doit penser à agrandir le champ de vision pour trouver l'endroit où se forme l'image : on laisse à l'élève l'occasion de réfléchir au

problème, plutôt que de lui faire subir la loi de l'ordinateur.

Au niveau de l'affichage des différents objets sur le banc, il faut évidemment que l'élève puisse les différencier : ceci se fera en associant à chaque objet une couleur spécifique. Il doit également pouvoir faire la distinction entre les deux types d'image (réelle, qui sera représentée par un trait continu et virtuelle, qui sera représentée par un trait en pointillé) et entre les deux types de lentilles (convergente et divergente, pour lesquelles on utilisera les conventions graphiques vu dans le paragraphe 2.4.1). En ce qui concerne la visualisation des rayons lumineux, on représentera les rayons réels par des traits continus et le prolongement de ces rayons (si c'est nécessaire) en pointillé.

La façon dont nous représentons le banc d'optique ne permet pas à l'étudiant de voir ce qui est projeté sur l'écran qu'il a placé sur le banc. On doit donc faire apparaître une fenêtre supplémentaire qui représentera ce qui est recueilli par l'écran. En fonction des positions de l'écran et de l'image, l'élève verra dans cette fenêtre ou bien rien, ou bien une image plus ou moins nette : l'image est nette lorsque l'écran se trouve là où l'image est formée et deviendra de plus en plus floue au fur et à mesure que l'écran s'éloigne de cette position pour finalement complètement disparaître.
Pour que l'élève puisse distinguer si l'image est à l'endroit ou à l'envers par rapport à l'objet optique, il faut naturellement qu'il y ait quelque chose qui distingue le haut du bas de l'objet. Pour ce faire, nous avons décidé que l'objet placé par l'élève sur le banc serait un objet en forme de flèche et que, dans le cas de l'objet, cette flèche pointerait toujours vers le haut. Ainsi si l'image de la flèche pointe vers le bas, ça veut dire que l'image est renversée par rapport à l'objet optique.

3.2.3 Equations

Pour cette partie-ci, l'interface va plutôt ressembler à un simple remplissage de formulaire : pour les différents objets, l'utilisateur va pouvoir introduire les valeurs des paramètres caractéristiques qu'il connaît. On devra aussi prévoir un espace pour afficher le résultat d'une résolution d'équation.

3.2.4 Exercices

Tout comme pour la partie relative aux équations, l'interface va être du style "remplissage de formulaire" : l'élève introduira ses réponses au fur et à mesure qu'il résoudra l'exercice. Pour chacun de ces exercices on affiche donc l'énoncé

et on prévoit une case à remplir pour chacune des réponses que l'élève est sensé fournir. On remarque évidemment que pour cette partie-ci et pour la partie "équations", l'aspect de l'interface est beaucoup moins important que pour la partie "simulation" où là l'aspect de l'interface est primordial.

3.3 Analyse fonctionnelle

3.3.1 Introduction

Nous commencerons cette analyse par décrire des fonctionnalités qui se reportent à l'entièreté du logiciel. Ensuite on analysera quelques fonctionnalités générales, c'est-à-dire qui sont relatives aux bancs d'optique proprement dit ou qui sont d'application pour les différents types d'objet qui peuvent être placé sur le banc. Finalement, nous parlerons des fonctionnalités qui sont spécifiques à chaque type d'objet.

3.3.2 Logiciel

Nous avons vu que notre logiciel peut être utilisé de deux façons différentes, à savoir le mode "Libre" et le mode "Problème". L'élève doit donc choisir la façon dont il veut utiliser le logiciel et peut naturellement passer d'un mode à l'autre en cours d'utilisation. Le passage d'un mode à l'autre ne passe pas de la manière identique dans les deux cas :

- Lorsque l'élève passe du mode "Libre" au mode "Problème", le programme efface tous les objets qui se trouvaient sur le banc, ainsi que toutes les données qui avaient été introduites dans la partie "équations". Ceci est tout à fait normal vu que l'élève va maintenant devoir résoudre un problème qui n'a rien à voir avec ce à quoi il était occupé auparavant.
- Lors du passage inverse, le logiciel n'efface rien. Cela laisse la possibilité à l'élève de continuer à travailler avec le montage qu'il a réalisé dans le cadre de l'exercice. Il peut ainsi approfondir un problème qu'il a rencontré lors de la résolution de cet exercice.

Ceci implique que l'élève a la possibilité de demander au logiciel de mettre tout à zéro, c'est-à-dire enlever tous les objets du banc et effacer toutes les données de la partie "équations", pour qu'il ne soit pas obligé de le faire lui-même.

L'aide en ligne, dans son état actuel, contient les informations suivantes :

- Explication des différents noms de variables utilisés : f, p, n, ...
- Conventions concernant les distances pour que l'étudiant sache bien quand il doit mettre un signe négatif ou un signe positif
- Brève explication concernant l'utilisation de la partie "simulation" et de la partie "équations".

3.3.3 Banc d'optique

Fonctionnalités générales

Les différents objets avec lesquels l'élève va réaliser son montage ont quelques fonctionnalités communes :

- L'objet en question peut être placé sur le banc d'optique.
- L'objet peut, par après, être également retiré du banc.
- L'élève a accès aux différents paramètres caractéristiques de chaque objet.
- Il peut aussi modifier ces paramètres.

En ce qui concerne la modification des paramètres caractéristiques, il y en a certains que l'élève ne peut modifier qu'à l'aide du clavier (c'est-à-dire en introduisant la nouvelle valeur) et il y en a d'autres pour lesquels il existe deux possibilités : introduire la nouvelle valeur ou faire la modification en agissant directement sur l'objet à l'aide de la souris. Nous reviendrons sur ce sujet lorsque nous décrirons les fonctionnalités spécifiques aux différents types d'objet.

On a déjà vu que le banc d'optique allait être muni d'une règle graduée et qu'on afficherait continuellement les coordonnées du curseur. L'élève pourrait donc utiliser ces deux moyens pour calculer la distance entre deux objets qui sont placés sur le banc. Mais l'élève dispose également d'une fonctionnalité "mètre ruban" qui lui permet de mesurer facilement et de façon très naturelle une distance horizontale entre deux objets.

Une fois que l'élève a disposé toutes les lentilles comme il le voulait et qu'il souhaite trouver la position de l'image, il peut procéder de deux façons différentes :

- Il peut demander au logiciel de lui montrer ce qui est projeté sur l'écran qu'il a mis sur le banc et ensuite bouger l'écran jusqu'à ce qu'il obtienne une image nette, signe qu'il se trouve à l'endroit où est formé l'image. De cette façon il peut par exemple vérifier si les calculs qu'il a effectués sont bien exacts.
- Il peut demander au logiciel de lui dessiner l'image. Cette méthode lui permet également de vérifier l'exactitude de ces calculs, mais ça lui évite surtout de devoir chaque fois rechercher l'image lorsqu'il modifie la valeur d'un paramètre dont il veut étudier l'influence.

Dans les deux cas, on veut que la simulation soit la plus réaliste possible : lorsque l'élève effectue un changement au niveau du montage, l'image va immédiatement et automatiquement s'adapter. Prenons l'exemple où l'élève a choisi de chercher

l'image à l'aide de son écran et supposons qu'il ait obtenu une image nette. Si maintenant il bouge horizontalement son objet ou une de ses lentilles, la position de l'image va naturellement aussi bouger et l'élève le verra sur son écran puisque l'image deviendra floue.

Afin de mieux comprendre comment l'image finale est obtenue, le logiciel peut montrer à l'élève les différentes images intermédiaires qui sont formées par les différentes lentilles. Dans ce cas, l'élève sélectionne la lentille pour laquelle il veut voir l'objet et l'image intermédiaires et peut donc parcourir ainsi une lentille après l'autre jusqu'à obtention de l'image finale.

Il est peut-être utile de remarquer que notre écran ne laisse pas passer les rayons lumineux. Cela veut donc dire que si on place l'écran à un endroit donné, C'est comme si toutes les lentilles qui se trouvent plus loin sur le banc n'y étaient pas : l'image calculée par l'ordinateur est celle formée par les lentilles qui se trouvent entre l'objet et l'écran.

Fonctionnalités relatives à l'objet "objet optique"

Le banc ne peut être muni au maximum que d'un seul objet optique. Cet objet optique est considéré comme étant une source de lumière. Le type de la source (monochromatique ou non) n'a pas d'importance vu que notre logiciel ne permet pas de visualiser l'aberration chromatique.

En ce qui concerne le déplacement de l'objet optique, on a un degré de liberté en plus par rapport aux autres types d'objet, car il peut être déplacé aussi bien horizontalement que verticalement. Ceci permet à l'élève de voir ce qui se passe lorsque l'objet est éloigné de l'axe optique.
La position est un de ces paramètres caractéristiques que l'élève peut modifier des deux façons, avec la souris ou en introduisant la nouvelle valeur.

L'élève peut également choisir la façon de modifier la hauteur de l'objet optique. Cette modification peut s'avérer fort utile lorsque l'on a un montage qui donne lieu à une image très grande ou très petite, puisqu'en changeant la taille de l'objet on change également celle de l'image.

Fonctionnalités relatives à l'objet "lentille"

Contrairement au cas de l'objet optique, l'élève peut placer plus d'une lentille sur le banc d'optique. Avant de pouvoir ajouter une lentille, il doit évidemment définir la lentille qu'il veut. Cela se fait en introduisant ou bien sa distance focale,

ou bien ses deux rayons de courbure et son indice de réfraction, puisque ces grandeurs sont liées entre elles par l'équation vue au paragraphe 2.4.2.

Concernant les paramètres caractéristiques de la lentille, il n'y a que la position que l'élève peut modifier avec la souris ou à l'aide du clavier. Les valeurs des autres paramètres (distance focale, indice de réfraction et rayons de courbure) peuvent uniquement être modifiées en tapant la nouvelle valeur au clavier.

Fonctionnalités relatives à l'objet "écran"

Tout comme pour l'objet optique, l'élève ne peut placer qu'un seul écran sur le banc.

L'écran ne possède qu'un seul paramètre caractéristique qui est sa position. Tout comme pour les deux autres types d'objet, ce paramètre peut être modifié des deux façons vues précédemment.

3.3.4 Equations

Ce volet-ci du logiciel permet à l'élève de représenter son montage de façon tout à fait numérique. Pour chaque montage on sait qu'on a un objet et une image, mais on ne sait pas combien de lentilles il y a. L'élève doit donc pouvoir ajouter (et naturellement aussi retirer) des lentilles pour réaliser convenablement la représentation numérique. Quelles dont les données qui peuvent être introduites ?

- Position de l'objet optique
- Position et nature de l'image formée par l'objet optique à travers les différentes lentilles, ainsi que l'agrandissement par rapport à l'objet
- Pour chaque lentille : la position, la distance focale, l'indice de réfraction, les rayons de courbure, la puissance, les distances p (i.e. la distance entre la lentille et ce qui lui sert d'objet) et p' (i.e. la distance entre la lentille et l'image intermédiaire qu'elle forme), ainsi que l'agrandissement de cette image intermédiaire.

Il est évident que l'élève ne doit pas introduire toutes ces données, mais il doit tout de même en donner un nombre suffisant pour que le système soit optiquement déterminé. Prenons l'exemple de la lentille : l'élève ne doit pas communiquer et la distance focale, et la puissance, et le triplet composé de l'indice de réfraction et de deux rayons de courbure. L'ordinateur n'a besoin que d'une de ces trois données afin de comprendre ce que l'étudiant veut obtenir. Si l'élève introduit maintenant plusieurs de ces valeurs et qu'il demande ensuite un transfert des données vers la partie "simulation", l'ordinateur va bien sûr d'abord contrôler la cohérence de toutes les données avant d'effectuer le transfert. Et dans le cas où il y a une incohérence (par exemple la distance focale ne correspond pas à la puissance de la lentille), le logiciel va signaler cette erreur à l'élève.

Lorsque l'élève veut connaître la valeur d'une donnée manquante, il peut la trouver en résolvant une des équations. Les équations qu'il a à sa disposition sont celles que nous avons vues dans les paragraphes 2.4.1, 2.4.2 et 2.4.3 :

$$\Pi = \frac{1}{f}$$
$$\frac{1}{f} = \frac{1}{f'} = (n_{1,2} - 1)(\frac{1}{R_1} - \frac{1}{R_2})$$
$$\frac{1}{f} = \frac{1}{p} + \frac{1}{p'}$$
$$M = \frac{i}{o}$$

En sélectionnant l'équation que l'utilisateur veut résoudre, on fait ressortir les paramètres qui interviennent dans l'équation pour que l'élève puisse voir s'il a bien introduit toutes les valeurs nécessaires (c'est-à-dire toutes sauf une, celle

qu'il veut trouver). Comme nous l'avons déjà signalé, la résolution se limite à celle d'une équation à une inconnue.

Quand le logiciel est utilisé en mode "Libre", l'élève dispose des deux fonctionnalités de transfert, "Simulation --→ équations" et "Equations → simulation", que nous avons déjà décrites dans le paragraphe 1.3.2. Rajoutons à cela que lors d'un transfert du banc vers la partie relative aux équations, l'ordinateur va calculer tous les paramètres de tous les objets et afficher leurs valeurs dans les cases adéquates.

Nous venons de voir que lorsque l'élève demande à l'ordinateur de faire un transfert des "équations" vers le banc d'optique, il y a automatiquement une vérification de la cohérence qui est faite. Cette fonctionnalité de vérification peut également être déclenchée explicitement par l'utilisateur. Ceci peut par exemple être intéressant lorsque le logiciel est utilisé en mode "Problème" où la fonctionnalité de transfert (et donc de vérification automatique) n'est pas disponible.
Il aurait aussi été possible de prévoir que cette fonctionnalité se déclenche automatiquement à chaque fois qu'une donnée est introduite ou corrigée, mais cela ne convient pas tout à fait. Prenons par exemple le cas où l'élève veut modifier deux données (R_1 et f). Il aurait à peine fait sa première modification que le logiciel lui signalerait qu'il y a une incohérence, alors que l'élève le sait très bien.

3.3.5 Exercices

L'élève peut choisir lui-même un exercice parmi l'ensemble des exercices contenus dans une base de données.

Une fois que l'élève a résolu l'exercice (en utilisant les fonctionnalités des parties "simulation" et "équations"), il peut demander au logiciel de corriger son exercice. La façon dont cette correction est effectuée a déjà été décrite au paragraphe 1.3.2.

Remarquons que les fonctionnalités liées à cette partie-ci du logiciel ne sont pas très nombreuses. Ceci est principalement dû au fait que c'est dans cette partie que sont situés la plupart des éléments que nous n'avons pas eu le temps d'implémenter.

Chapitre 4

Analyse conceptuelle

4.1 Modèle d'architecture en niveaux

Le modèle que nous allons utiliser pour faire l'analyse conceptuelle est le modèle d'architecture vu au cours de Méthodologie de Développement de Logiciels [Dub98]. Dans ce modèle, on découpe le logiciel en différents modules que l'on place ensuite dans les niveaux appropriés.

4.1.1 Découpe en modules

La modularisation d'un programme possède trois grandes caractéristiques qui sont le "information hiding", le couplage et la cohésion.

"Information hiding"

Chaque module va être divisé en deux parties : son interface et son corps. La partie "interface" d'un module est la partie qui est visible pour tous les autres modules du logiciel. C'est donc là qu'on fait connaître aux autres modules quels services leur sont offerts par le module en question. La partie "corps" n'est accessible qu'au module proprement dit. C'est à cet endroit qu'on place ce qu'on désire cacher aux autres : les structures des données intermédiaires, la manière dont sont implémentés les services, ...

Couplage

Le couplage est la relation qui indique l'utilisation de services par d'autres modules. Ainsi, on a par exemple que dans le corps du module B, on utilise des services proposés respectivement par les modules A et C (fig 4.1).

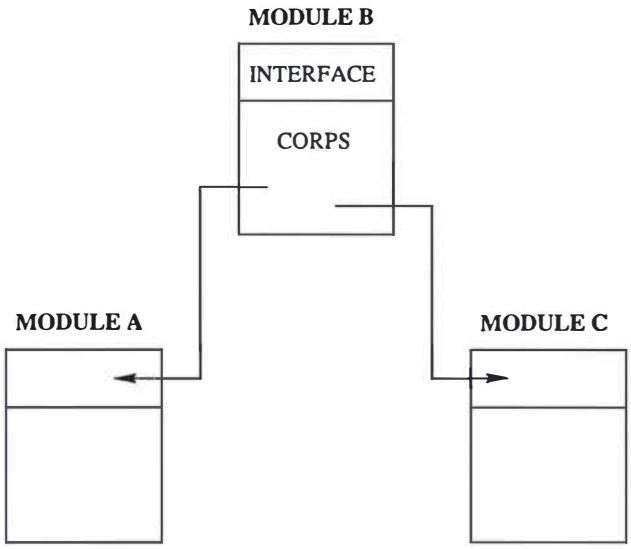

FIG. 4.1: Couplage entre le module B et les modules A et C.

Le cas le plus mauvais est celui où tous les modules utilisent les services de tous les autres modules. C'est ce qu'on appelle le couplage maximal : tout le monde dépend de tout le monde.
Il est préférable d'essayer d'aller vers un couplage minimal. Pour nous aider dans cette recherche, nous allons adopter une architecture en niveaux, décrite dans le paragraphe 4.1.2.

Cohésion

La cohésion est le principe sur lequel on se base pour savoir quels services vont être regroupés dans le même module. Il existe cinq différents types de cohésion :

- *Cohésion accidentelle* : il n'y a aucune logique qui dicte la façon de regrouper les services. Le regroupement se fait au hasard.
- *Cohésion temporelle* : on met ensemble les processus qui se déroulent en même temps. On obtient ainsi un regroupement qui est lié à l'exécution du programme.
- *Cohésion liée a la logique du problème* : on va par exemple regrouper toutes les fonctionnalités qui s'occupent de la gestion d'erreur.
- *Cohésion fonctionnelle* : il peut arriver que plusieurs services utilisent des sous-fonctionnalités communes. Plutôt que de réécrire ces procédures identiques à plusieurs endroits, on choisi de les rassembler dans des modules qui sont accessibles à ceux qui en ont besoin.

- *Cohésion informationnelle* : on regroupe les services qui portent sur la même structure de données, qui en programmation orientée-objet, pourra être représentée par un objet (au sens informatique du terme).

4.1.2 Principe d'architecture en niveaux

Dans ce qui précède, nous avons vu qu'un des problème rencontré lors de la modularisation du logiciel est le problème du couplage. Dès que le nombre de modules devient un peu élevé, il y a beaucoup de chance qu'on obtienne un réseau complexe de liens d'utilisation entre les modules. Pour remédier a ce problème, nous allons adopter l'architecture en niveaux vue au ours de Génie Logiciel [Dub98], pour laquelle on impose les contraintes suivantes :

- Un module ne peut appartenir qu'à un seul niveau.
- Pour chaque niveau, on a 0, 1 ou plusieurs modules.
- Un module d'un niveau donné peut utiliser uniquement les services proposés par les modules de niveau égal ou inférieur au sien.

Cette architecture est composée de six niveaux. Nous allons les parcourir du niveau le plus haut vers le niveau le plus bas :

- *Niveau 6* : Ici se situent les modules dits coordinateurs, c'est-à-dire ceux qui permettent l'enchaînement des différentes fonctionnalités. Dans la plupart des cas, on a plusieurs modules coordinateurs.
- *Niveau 5* : C'est le niveau contenant les modules qui gèrent l'aspect interface homme-machine du logiciel (affichage des fenêtres de dialogue, menus, ...).
- *Niveau 4* : Dans ce niveau, on retrouve tous les modules dans lesquels on effectue du traitement. C'est la partie algorithmique du logiciel.
- *Niveau 3* : Ce niveau est formé des modules qui s'occupent des données persistantes du logiciel (par exemples les données contenues dans une base de données).
- *Niveau 2* : Ce niveau est aussi appelé le niveau "middleware". Il comporte tous les outils et les services qui sont offerts par d'autres logiciels du système, comme par exemple les services d'accès à une base de données Interbase.
- *Niveau 1* : C'est le niveau du système d'exploitation qui offre aussi un certain nombre de fonctionnalités de base.

4.2 Architecture logique

4.2.1 Modularisation de notre projet

A la figure 4.2, nous avons représenté graphiquement l'architecture logique de notre logiciel. Notons que dans cette représentation, nous avons volontairement omis le niveau Interface Homme-Machine pour des raisons de clarté. Les liens entre le niveau Coordinateur et le niveau Traitement passent donc en réalité par des modules IHM.
Nous nous sommes également limités aux trois niveaux supérieurs. Nous avons décidé de ne pas détailler les niveaux inférieurs (niveaux 1 et 2), non pas que nous n'utilisons pas leurs services, mais étant donné que nous n'avons pas implémentés ces modules nous-mêmes, nous supposons implicitement leur existence et l'utilisation de leurs services.
Nous allons maintenant analyser ces différents modules : quel est leur type de cohésion, quels services offrent-ils, comment explique-t-on leurs relations d'utilisation ?

4.2.2 Modules "Vue-Exer", "Vue-RepNum" et "Vue-RepSim"

Schéma Entité-Association

Afin de pouvoir définir la structure de la base de données qui contient les exercices de notre logiciel, nous nous sommes basés sur son schéma ERA (voir figure 4.3).
Nous avons donc que chaque exercice (EXERCICE) est caractérisé par un numéro, qui est son identifiant, et par un énoncé.
Nous avons vu que la résolution d'un exercice par un élève est une résolution double : l'élève doit trouver les valeurs numériques de certains paramètres et réaliser le montage correspondant à l'énoncé. Pour que le logiciel puisse faire la correction de ces deux ensembles de réponses, il doit naturellement pouvoir accéder aux différents résultats corrects. Dans notre schéma ERA, ceci se traduit par deux associations qui sont rattachées à l'entité EXERCICE.
Premièrement, nous avons qu'à un exercice donné correspond un certain nombre de réponses numériques (REPONSE-NUM). Une réponse numérique est caractérisée par un label qui exprime le type de la réponse attendue : distance focale d'une lentille, position de l'image, agrandissement de l'image, ... Elle est également caractérisée par la valeur numérique de la réponse et l'unité de mesure dans laquelle on exprime la valeur.
Deuxièmement, on fait correspondre autant de REPONSE-SIM qu'il n'y a de

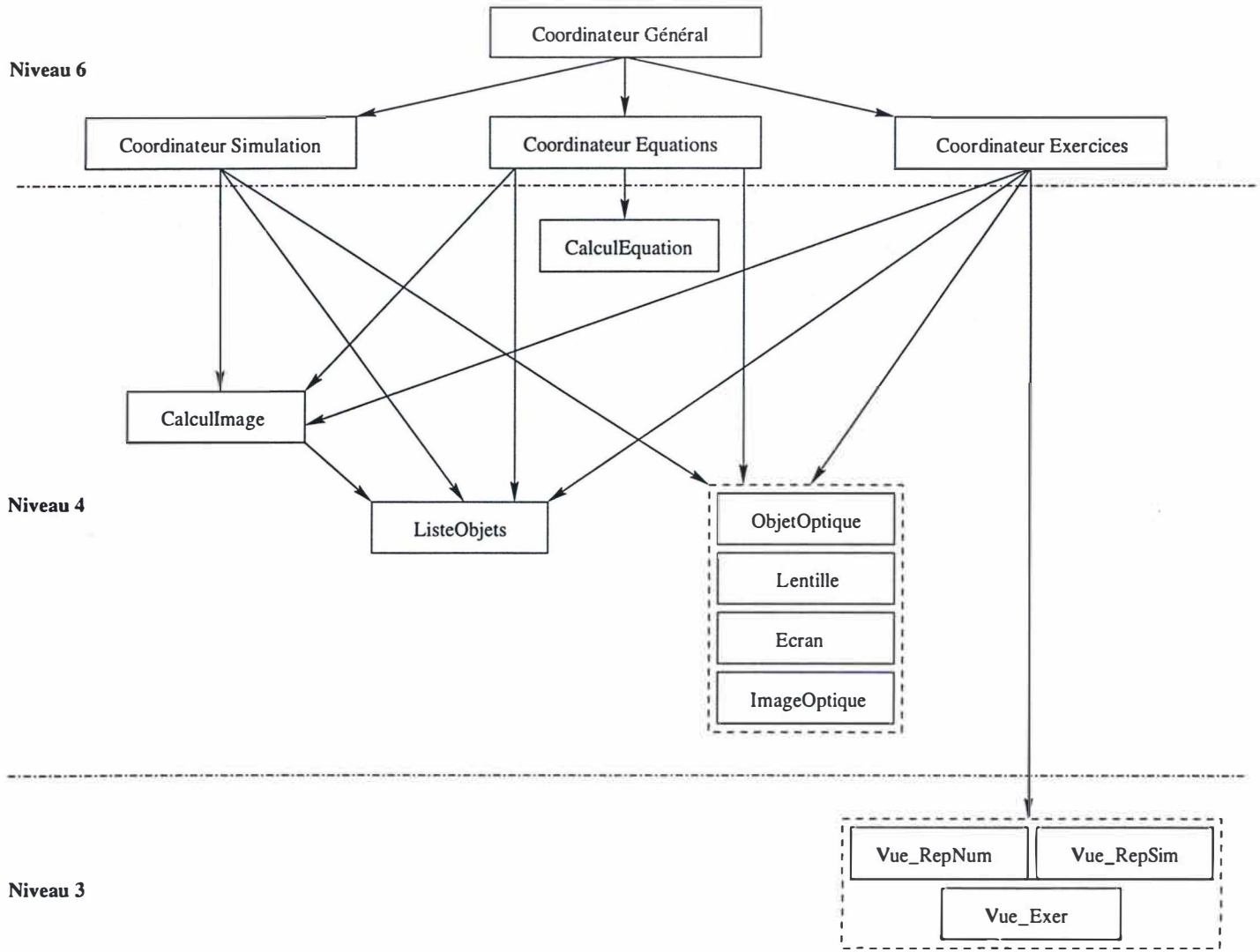

FIG. 4.2: Architecture logique du logiciel.

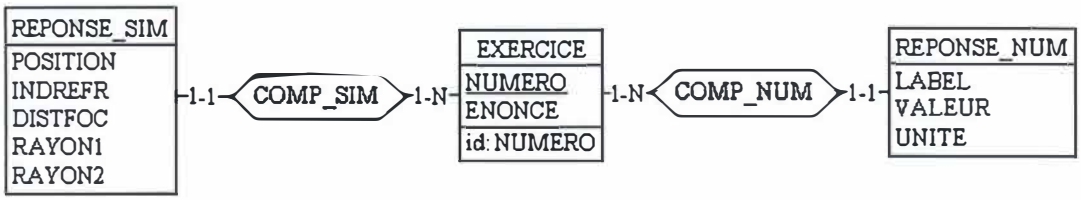

FIG. 4.3: Schéma Entité-Association.

lentilles dans le montage correct. Chacune de ces réponses relatives à l'aspect simulation sont donc caractérisées par la position de la lentille par rapport à l'objet, par leur distance focale, par leur indice de réfraction et par leur deux rayons de courbure. A partir de ces données, l'ordinateur pourra calculer la position de l'image.

Différentes Vues

Lorsqu'on transforme notre schéma ERA en un schéma physique, on obtient une base de données qui est composée de trois tables, chaque table correspondant à une entité du schéma. A chacune de ces tables, on va associer des modules séparés, qui créent des vues partielles du schéma de base et qui contient les opérations que l'on peut faire sur ces vues. Ainsi, on obtient que dans un module donné, on regroupe tous les services qui s'appliquent à une même structure de données. Ces modules présentent donc une cohésion de type informationnelle (cohésion orientée-objet).
Dans le module Vue-Exer, on regroupe les services qui permettent d'accéder à l'énoncé d'un exercice identifié par son numéro, d'ajouter l'énoncé d'un nouvel exercice.
Les modules Vue-RepNum et Vue-RepSim offrent les services qui, pour un exercice particulier, donnent respectivement accès aux réponses numériques et aux données nécessaires pour vérifier le montage. On y retrouve également les services permettant d'ajouter les différentes réponses lorsqu'on ajoute un exercice.

4.2.3 Modules "ObjetOptique", "Lentille", "Ecran", et "Image"

Nous avons regroupé l'analyse des quatre modules dans le même paragraphe car ils présentent beaucoup de similitudes au niveau des services offerts. Ces quatre modules sont tous des modules à cohésion informationnelle. A chaque module on fait correspondre un des objets que l'élève peut placer sur le banc :

objet optique, lentille et écran. Il y a également un module pour l'image, bien que ce ne soit pas un objet sur lequel l'élève peut agir directement (comme c'est le cas pour les trois autres). L'existence de ce module se justifie simplement par le fait qu'il existe des services qui portent sur la structure des données qui représentent l'image.

Chacun de ces objets va avoir une structure de données assez similaire, dont une partie sera même identique, car ils ont un paramètre caractéristique en commun, la position horizontale sur le banc. Pour le reste, on complète cette structure en ajoutant, pour chaque objet, les paramètres caractéristiques qui lui sont propres :

- *Objet optique* : sa taille et sa position verticale.
- *Lentille* : sa distance focale, son indice de réfraction et ses deux rayons de courbure.
- *Ecran* : /
- *Image optique* : sa taille, sa nature, son sens (droite / renversée) et sa position verticale.

Au niveau des services offerts, chaque module offre les trois mêmes services :

- Création de l'objet.
- Destruction de l'objet.
- Affichage visuel de l'objet.

4.2.4 Module "ListeObjets"

Ce module-ci va s'occuper de la gestion dynamique de l'ensemble des objets qui composent le montage réalisé sur le banc. Ceci se fait à l'aide du type abstrait "ListeObjets" qui est une liste contenant les différents objets du montage, triés selon leur position horizontale sur le banc. Dans ce module, nous offrons aux autres modules toute une série de service utiles qui vont agir sur cette structure de données. On a donc affaire à un module à cohésion informationnelle.

Quels sont les services proposés par ce module ?

- Création et destruction de la liste
- Tri de la liste : les objets de la liste sont triés par position horizontale croissante.
- Ajout ou suppression d' un objet de la liste.

- Recherche dans la liste :il y a différents types de recherche possibles selon les renseignements que l'on veut obtenir : nombre de lentilles sur le banc, présence d'un certain type d'objet sur le banc, présence d'un objet à une certaine position sur le banc, ...

Il est évident qu'il y a encore d'autres services qui sont disponibles tels que : compter le nombre d'objets dans la liste, prendre le premier élément de la liste, prendre l'élément suivant, ... Mais il s'agit de services de base qui sont délivrés par le niveau "middleware".

4.2.5 Module "CalculImage"

Dans ce module, on retrouve les services qui s'occupent des calculs relatifs à la formation de l'image.
Le module propose trois services :

- Le premier service offert est celui qui calcule l'image finale formée par l'ensemble des lentilles. Avant d'exécuter les calculs, on commence par trier la liste des objets. Ensuite, on calcule successivement l'image formée par chacune des lentille pour aboutir enfin à l'image finale, munie de tous ses paramètres caractéristiques. Pour obtenir ce résultat, on fait appel à des services du module "ListeObjets".
- Le deuxième service permet à l'élève de visualiser pour une lentille donnée, l'objet et l'image intermédiaire, ainsi que les rayons qui forment cette dernière. Pour obtenir ce résultat, on procède de la même façon que pour le service précédent, sauf qu'à présent, on affiche pour la lentille voulue son objet et son image intermédiaire ainsi que les rayons qui ont permis de déterminer la position de cette image intermédiaire.
- Le dernier service est celui qui, en fonction de la position de l'écran et de l'image finale, va calculer la forme de l'image qui est observée sur l'écran (placé sur le banc).

La cohésion de ce module est fonctionnelle, car on y a regroupé des services qui sont utilisés à plusieurs reprises par d'autres modules.

4.2.6 Module "CalculEquations"

Ce module va contenir tous les services qui effectuent les calculs relatifs à l'aspect "équations" du logiciel. Il s'agit d'un module à cohésion fonctionnelle car

ses services vont être utilisés à différents endroits dans le module "Coordinateur Equations".

Quels sont les services offerts par ce module et quels sous-services utilise-t-il pour y arriver ?

- Résolution d'une équation à une inconnue : cette fonctionnalité-ci n'utilise pas d'autres sous-fonctionnalités. C'est un calcul qui s'effectue uniquement sur base des données introduites par l'élève.
- Vérification de la cohérence : tout comme pour la fonctionnalité précédente, on ne fait que du calcul qui implique les valeurs que l'élève a introduites. Aucun service d'un autre module n'est nécessaire.

4.2.7 Module "Coordinateur Exercices"

Ce module s'occupe de coordonner les différents services qui sont nécessaires au bon déroulement de l'aspect "exercices" de notre logiciel. Cela va faire intervenir des services des niveaux IHM, traitement et données persistantes.

Pour ce qui est du niveau IHM, le coordinateur va demander en temps voulu l'affichage et l'effacement des fenêtres de dialogue qui vont guider l'élève dans le choix, dans la résolution et dans la correction d'un exercice.

Lorsque l'élève parcourt la liste des exercices possibles, le coordinateur doit veiller à lui présenter le bon énoncé. A cette fin, il utilise le service du module "Vue-Exer" qui lui renvoie l'énoncé de l'exercice sélectionné par l'élève.

Lors de la correction d'un exercice, le module va utiliser des services qui sont livrés par les modules "Vue-RepSim" et "Vue-RepNum" et "ListeObjets".
Dans la première phase de correction (voir paragraphe 1.3.2), le logiciel se limite à dire si la réponse est correcte ou non. il commence donc par comparer les valeurs introduites par l'élève avec les valeurs attendues qu'il obtient grâce au service offert par le module "Vue-RepNum". La partie vérification de la partie "simulation" est plus compliquée, car il peut arriver que l'élève n'ait pas mis le bon nombre de lentilles ou qu'il ait oublié de placer l'écran. Pour vérifier les erreurs de ce type-là, on utilise les services du module "ListeObjets" : on vérifie un à un si tous les objets ont été placés sur le banc, et si le nombre de lentilles est exact. Dès qu'on a vérifié que la "composition" du montage est correcte, on peut contrôler si les différentes composantes se trouvent au bon endroit. Les positions et les paramètres corrects nous sont délivrés par le biais du module "Vue-RepSim".

Ces différents services vont également nous être utiles lorsque l'élève veut passer à la deuxième phase de la correction, qui précise les endroits où il y a erreur. Dans le cas où l'élève demande au logiciel de lui afficher la bonne solution (troisième phase), on utilise encore d'autres services des modules des différents objets et du module "ListeObjets". On doit pouvoir créer de nouveaux objets ou bien supprimer ceux dont on n'a plus besoin. Et il faut également pouvoir les ajouter dans la liste ou les en retirer. De cette façon, l'élève peut visualiser le montage qu'on attendait de lui.

4.2.8 Module "Coordinateur Equations"

Ce module coordinateur va s'occuper de l'enchaînement des services utilisés par la partie "équations" du programme, de sorte que l'élève puisse représenter numériquement son montage, résoudre des équations pour trouver les valeurs voulues, vérifier la cohérence des données numériques et effectuer les deux types de transfert.

Tout comme c'était le cas pour le "Coordinateur Exercices", ce coordinateur-ci va également faire appel aux modules du niveau IHM lorsque cela est nécessaire : affichage de la fenêtre où l'élève peut introduire les valeurs souhaitées, affichage des messages concernant la cohérence ou incohérence des données, ...

Lorsque l'élève veut résoudre une équation ou vérifier la cohérence de sa représentation, le coordinateur va utiliser les services du module "CalculEquations" qui prendront en charge les calculs relatifs à ces actions.

Analysons ce que le coordinateur doit faire lors des opérations de transfert :

- *Transfert "Equations → Simulation"* : Ici, on commence par vérifier la cohérence des données en utilisant le service approprié qui se trouve dans le module "CalculEquations". Une fois la cohérence contrôlée, on doit réaliser le montage correspondant à la demande de l'élève. On doit donc pouvoir créer et détruire des objets ainsi que les intégrer ou les retirer de la liste, ce qui peut être fait par le biais des services adéquats offerts par les modules des objets respectifs et par le module "ListeObjets". Les calculs concernant l'image sont effectués par le module "CalculImage".
- *Transfert "Simulation → Equations"* : Dans ce cas-ci, le seul service utilisé provient du module "CalculEquations" : certaines valeurs numériques comme par exemple la puissance d'une lentille ne sont pas directement accessibles, mais doivent être calculées en résolvant les équations.

4.2.9 Module "Coordinateur Simulation"

Ce coordinateur propose tous les services qui sont relatifs à la simulation du banc d'optique. Comme nous l'avons déjà vu, l'élève dispose d'un grand nombre de fonctions concernant le simulateur proprement dit :

- Placer / déplacer / enlever un objet.
- Déplacer verticalement l'objet optique.
- Agrandir / rétrécir l'objet optique.
- Faire afficher les données caractéristiques d'un objet.
- Visualiser ce qui est recueilli sur l'écran.
- Visualiser l'image finale / une des images intermédiaires.
- Effectuer un zoom horizontal sur le banc.
- Mesurer une distance horizontale entre deux points.

Il est évident que pour la plupart de ces fonctions, le coordinateur va utiliser les services des modules du niveau IHM.
Pour ce qui est de la visualisation de l'image, le coordinateur passe naturellement par l'intermédiaire du module "CalculImage".

4.2.10 Module "Coordinateur Général"

Au niveau de ce module-ci, on va gérer les services qui englobent tout le logiciel : choix du mode de fonctionnement et "Mise a zéro". Pour mener ces opérations à bien, le coordinateur va faire appel à ses sous-coordinateurs.

Lorsque l'élève a envie de recommencer quelque chose depuis le début, le "Coordinateur Général" va demander aux autres coordinateurs de s'occuper de la mise à zéro de leur partie : Le "Coordinateur Simulation" enlève tous les objets du banc, le "Coordinateur Equations" efface toutes les données qui avaient été introduites et le "Coordinateur Exercices" fait de même en ce qui concerne les réponses que l'élève aurait déjà données.

Concernant le choix du mode de fonctionnement, le "Coordinateur Général" va également se baser sur des services offerts par ses sous-coordinateurs. Dans le cas du mode "Libre", il faut que l'élève ait la possibilité d'effectuer les deux types de transfert et on ne doit pas afficher la fenêtre relative aux exercices. Tandis que dans le mode "Problème", on doit naturellement afficher cette fenêtre, mais par contre, l'élève n'a plus la possibilité de transférer des données d'une partie à l'autre.

Chapitre 5

Implémentation

5.1 Introduction

Lors de l'analyse conceptuelle nous avions "construit" un certain nombre de modules. Comment va-t-on retrouver ces modules dans notre implémentation ? Les modules à cohésion informationnelle vont donner lieu à la création d'objets[1], tandis que pour les modules à cohésion fonctionnelle on va voir apparaître des fonctions et des procédures. Nous avons donc choisi d'implémenter notre simulateur didactique en orientée-objet, et plus particulièrement en Delphi [Can96].

Dans ce chapitre nous allons expliquer comment nous avons implémenté les fonctions/procédures, les objets et leurs services offerts, ainsi que la façon dont nous avons conçu l'interface du logiciel. Pour pouvoir distinguer le pseudo-code du reste du texte , nous avons pris comme convention d'utiliser cette `police-ci` pour le pseudo-code.
Le listing complet du programme se trouve en annexe.

[1]Pour ne pas confondre les objets informatiques avec les objets que l'on place sur le banc, nous désignerons ces derniers par le terme "composants".

5.2 Objets relatifs aux composants du banc

5.2.1 Objet "TComp"

Structure

Lors de l'analyse conceptuelle nous avons remarqué que les différents composants du banc possèdent une caractéristique commune qui est leur position horizontale sur le banc. De plus, nous avons que la cohésion des modules correspondants à ces composants est informationnelle, c'est-à-dire orientée-objet. C'est pour cela que nous avons décidé de créer un objet générique "TComp" dont héritent les objets associés aux composants. La structure de cet objet générique est la suivante :

```
TComp=class(TGraphicControl)
  TypeComp : TTypeComp ;
  Position : real ;
end ;
```

L'objet TComp est un objet qui découle de l'objet TGraphicControl (fourni par Delphi), grâce auquel on dispose de la propriété "Canvas" (canevas) qui nous procure une surface dans laquelle on va pouvoir dessiner à l'aide de la méthode Paint. Nous verrons comment cela se passe concrètement lorsque nous décrirons les objets associés aux composants.

Propriétés

Pour obtenir notre objet générique TComp, nous avons ajouté deux nouvelles propriétés à celles qui existaient déjà :

- *TypeComp* : Cette propriété peut prendre une des quatre valeurs que nous avons définies dans le type énuméré TTypeComp. A chacune de ces valeurs correspond un type de composant : objet optique, image optique, lentille ou écran. Nous utilisons cette propriété lorsque nous parcourons la liste des composants pour vérifier la présence d'un certain type de composant sur le banc.
- *Position* : Cette propriété contient la position horizontale du composant sur le banc.

5.2.2 Objet "TObjetOpt"

Structure

Cet objet correspond à l'objet optique de notre banc. Comme nous l'avons expliqué dans le paragraphe précédent, cet objet descend de l'objet TComp. Nous avons défini sa structure comme suit :

```
TObjetOpt=class(TComp)
  PosY : integer ;
  Real : boolean ;
  Up : boolean ;
  Point : real ;
  Pen : TPen ;
  constructor Create (AOwner : TComponent) ; override ;
  destructor Destroy ; override ;
  procedure Paint ; override ;
  procedure Resize ;
end ;
```

Remarquons que nous utilisons également cet objet TObjetOpt pour représenter l'image optique. Dans le restant de ce paragraphe, nous parlerons toujours de l'objet optique, mais il ne faut pas perdre de vue que tout ce qui d'application pour l'objet optique, l'est également pour l'image.

Propriétés

Les propriétés supplémentaires dont nous avons garni cet objet sont les suivantes :

- *PosY* : contient la position verticale de l'objet optique, exprimée par rapport à l'axe optique.
- *Real* : exprime si l'objet optique est réel ou virtuel.
- *Up* : nous renseigne sur le sens de l'objet optique (droit ou renversé).
- *Point* : cette propriété nous donne la position verticale de la pointe de notre objet optique (qui, rappelons le, est une flèche). Lorsque nous montrons comment les images intermédiaires sont obtenues, nous affichons toujours des rayons qui sont émis par ce point-là de l'objet optique.
- *Pen* : pour des raisons de clarté, nous avons choisi de ne pas dessiner l'objet optique et l'image dans la même couleur. De plus, nous faisons également la distinction entre une image réelle ou virtuelle en changeant le style du trait utilisé. L'objet TPen est l'objet utilisé pour dessiner des lignes au sein

d'un canevas. C'est une sorte de "stylo", dont on peut définir la couleur, l'épaisseur, le style de trait, ... En ajoutant cette propriété à la structure de l'objet, nous pouvons associer à l'objet optique et à l'image un "stylo" différent.

Méthodes

- *Constructor Create* : crée et initialise le nouvel objet et l'insère dans son propriétaire spécifié par le paramètre AOwner.
  Dans la déclaration de cette méthode, nous avons ajouté la commande override : ceci nous permet de raffiner la méthode héritée, plutôt que de la remplacer. Ce raffinement n'est possible qu'avec des méthodes virtuelles, tel que Create et implique que l'on doit d'abord appeler la méthode héritée.

  ```
  Appeler la méthode Create héritée de TComp.
  Créer l'objet Pen.
  Modifier les propriétés du stylo.
  Définir le type de composant.
  Définir la hauteur et la largeur de l'objet graphique.
  Associer des méthodes aux actions qui peuvent être
  effectuées sur l'objet optique à l'aide de la souris
  (cliquer,double-cliquer, ...).
  ```

- *Destructor Done* : détruit l'objet et libère la mémoire qu'il utilisait. Vu que nous allons allouer des ressources pour l'objet Pen dans notre objet Pen, nous sommes obligés de raffiner cette méthode-ci pour libérer ces ressources.

  ```
  Définir stylo.
  Appeler la méthode Destroy héritée de TComp.
  ```

- *Procedure Paint* : dessine au sein du canevas le dessin correspondant à l'objet optique. Ici aussi nous raffinons la méthode Point héritée (qui initialise le canevas).

  ```
  Associer au stylo du canevas le stylo défini dans Pen.
  Dessiner la ligne qui représente l'objet graphique.
  ```

- *Resize* : calcule la nouvelle position de l'objet graphique au sein de son propriétaire. Cette méthode est utilisée lorsqu'on change, par exemple, la taille de la fenêtre (son propriétaire) dans laquelle on a dessiné l'objet optique.

```
Calculer positions horiz. et vert. de l'objet graphique.
```

5.2.3 Objet "TLens"

Structure

La représentation d'une lentille sera faire par l'objet TLens qui hérite de l'objet TComp. Quelle est sa structure ?

```
TLens=class(TComp)
  Focus : real ;
  Radius1 : real ;
  Radius2 : real ;
  RefrInd : real ;
  PosObj : real ;
  PosIm : real ;
  Agr : real ;
  LNumber : integer ;
  constructor Create (AOwner : TComponent) ; override ;
  procedure Paint ; override ;
  procedure Resize ;
end ;
```

Propriétés

- *Focus, Radius1, Radius2 et RefrInd* : ce sont tous les paramètres qui caractérisent une lentille. Nous n'y avons pas ajouté la puissance, car elle peut très facilement être calculée en prenant l'inverse de la distance focale.
- *PosObj, PosIm, et Agr* : ce sont trois propriétés qui sont utilisées lorsqu'on calcule l'image formée par l'ensemble des lentilles qui ont été placées sur le banc. Chacune de ces propriétés contient respectivement la position de ce qui sert d'objet à cette lentille , la position de l'image intermédiaire formée par cette lentille et l'agrandissement de cette image.
- *LNumber* : on associe un numéro à chaque lentille qui est placée sur le banc par l'élève. Ce numéro nous sera utile lors du transfert du banc vers la partie équation et lors du transfert inverse.

Méthodes

Les méthodes que nous avons implémentées pour cet objet sont les mêmes que pour l'objet TObjetOpt, sauf qu'ici nous ne devons pas raffiner le destructeur car

on n'alloue pas de ressources dans le constructeur Create.

- *Constructor Create* : le déroulement de cette méthode est identique à celui vu pour l'objet TObjetOpt .
- *Procedure Paint* : concernant le dessin d'une lentille, on doit faire la distinction entre une lentille convergente et divergente.

```
Dessiner la droite commune aux deux types de lentille.
Si distance focale positive
  Alors dessiner flèches pointant vers l'extérieur,
  Sinon dessiner flèches pointant vers l'intérieur.
Fin si.
Dessiner les deux foyers de la lentille.
```

- *Resize* : cette méthode se déroule de la même façon que pour l'objet TObjetOpt.

5.2.4 Objet "TScreen"

Structure

Cet objet correspond au composant "écran" que l'utilisateur peut placer sur le banc. Comme nous l'avons déjà décrit précédemment, l'objet TScreen est un fils de l'objet TComp. Sa structure se présente sous la forme suivante :

```
TScreen=class(TComp)
  constructor Create (AOwner : TComponent) ; override ;
  procedure Paint ; override ;
  procedure Resize ;
end ;
```

Méthodes

Les méthodes sont les mêmes que celles de l'objet précédent et leur déroulement est identique.

5.3 Objet relatif à la liste des composants

Structure

Dans le chapitre consacré à l'analyse conceptuelle, nous avons parlé du fait que nous rassemblons tous les composants du banc dans une liste triée, ce qui nous facilite un certain nombre d'opérations : calcul de l'image, transfert des données du banc vers la partie "équations", ... Cette liste va être concrétisée à l'aide de l'objet TListeComp, qui est un descendant de l'objet TList fourni par Delphi.

```
TListeComp=class(TList)
  procedure TriComp ;
  function FindComp(CompPos : real ; CompType : TTypeComp) : integer ;
  function FindTypeComp(CompType : TTypeComp) : integer ;
  function FindLensNum(number : integer) : integer ;
  function CountLens : integer ;
  function FindNextLens(last : integer) : integer ;
end ;
```

Méthodes

Sur l'objet TList on peut effectuer toute une série d'opérations de base telles que pointer vers le premier/dernier élément, compter le nombre d'éléments, pointer vers l'élément suivant, ... A toutes ces méthodes élémentaires, nous avons encore ajouté les suivantes :

- *Procédure TriComp* : se base sur la méthode Sort héritée de TList, qui prend comme argument une fonction qui fournit la manière dont doit être effectué le tri (ici, ordre croissant des positions horizontales).
- *Function FindComp* : permet de vérifier l'existence, et de trouver la position dans la liste, d'un certain type de composant à une position donnée.

```
Résultat ← -1.
Aller au premier élément de la liste.
Tant que pas fin de liste et résultat = -1
  Faire
    Si type objet et positions corrects
      Alors résultat ← position élément dans la liste.
    Fin si.
    Aller à l'élément suivant.
  Fin faire.
Renvoyer résultat.
```

Le déroulement des méthodes suivantes est chaque fois très semblable à celui qu'on vient de décrire. C'est pourquoi nous ne le détaillerons pas pour les autres. Le résultat renvoyé par la fonction est tout le temps la position dans la liste de l'élément recherché et vaut -1 si l'élément n'a pas été trouvé.

- *Function FindTypeComp* : cherche dans la liste un composant d'un type donné.
- *Function FindLensNum* : permet de trouver la lentille qui possède le numéro donné.
- *Function CountLens* : compte le nombre de lentilles présentes dans le montage et dont on doit tenir compte lors du calcul de l'image, c'est-à-dire celles qui se trouvent entre l'objet optique et l'écran.
- *Function FindNextLens* : trouve la lentille suivante du montage.

5.4 Procédures et fonctions de calcul

5.4.1 Calcul de l'image

Comme nous l'avons déjà signalé précédemment, nous déterminons l'image finale en calculant une à une les images intermédiaires qui sont formées par les différentes lentilles.
Pour trouver la position de chacune des images intermédiaires, nous utilisons la construction géométrique vue au paragraphe 2.4.3, c'est-à-dire que nous calculons l'intersection entre deux rayons réfractés. Ce point d'intersection nous donne les positions horizontale et verticale du point qui émet les rayons utilisés pour le calcul (i.e. la pointe de la flèche qui sert d'objet). Pour connaître la position verticale de l'autre extrémité de l'image, il suffit de calculer l'agrandissement de l'image et d'en déduire la position recherchée.
Dans cette procédure nous calculons déjà toutes les images intermédiaires. C'est pour cette raison que nous avons décidé d'utiliser cette même procédure pour l'affichage de l'image intermédiaire formée par une lentille donnée. A chaque fois que nous traitons le cas d'une lentille, nous vérifions si l'utilisateur désire visualiser l'image intermédiaire formée par cette lentille. Si c'est le cas, alors nous dessinons à l'écran ce qui sert d'objet à cette lentille, l'image intermédiaire formée par cette lentille, ainsi que deux des rayons qui la forment (ceux qui sont utilisés pour la construction géométrique). A propos de ces rayons, nous faisons la distinction entre les rayons proprement dits et leur prolongement en dessinant les premiers en trait continu et les seconds en pointillé.

```
Trier la liste des composants.
numLens ← nombre de lentilles dont on doit tenir compte.
lensPos ← première lentille située à droite de l'objet.
Si numLens > 0
  Alors
    Image intermédiaire ← objet optique du banc.
    Pour i=1 à numLens
      Faire
        Objet intermédiaire ← image intermédiaire.
        Calculer positions et agrandissement de l'image
        intermédiaire.
        Affecter les valeurs aux champs PosObj, PosIm et Agr
        de la lentille concernée.
        Calculer autres paramètres de l'image intermédiaire
        (pos. vert., sens et nature) en fonction de
        l'agrandissement.
        Si bouton "Image intermédiaire" est enfoncé et
        si i=lentille sélectionnée par l'utilisateur
          Alors
            Dessiner objet et image intermédiaires.
          Fin si.
        Trouver lentille suivante.
      Fin faire.
    Image finale ← image intermédiaire.
    Définir style de trait du stylo de l'image.
  Fin si.
```

5.4.2 Vérification de la cohérence des données numériques

```
Pour toute lentille de la partie "équations"
  Faire
    Si valeurs ont été introduites pour distance focale
    et puissance
      Alors vérifier cohérence entre les deux.
    Si valeurs ont été introduites pour distance focale et
    triplet (n,R1,R2)
      Alors vérifier cohérence entre les deux.
    Si valeurs ont été introduites pour puissance et
    triplet (n,R1,R2)
      Alors vérifier cohérence entre les deux.
  Fin faire.
Pour toute lentille de la partie "équations"
  Vérifier si valeurs ont été introduites pour position
  et pour distance focale (ou puissance, ou (n,R1,R2).
Pour toute lentille de la partie "équations"
  Faire
    Calculer valeurs de p, p' et agr.
    Vérifier valeurs introduites pour p, p' et agr.
  Fin faire.
```

Si on détecte une erreur dans une des vérifications, on en avertit l'utilisateur et on arrête le processus de vérification.

5.4.3 Résolution d'une équation

```
En fonction de l'équation sélectionnée
et en fonction de la donnée manquante
  Calculer valeur de la donnée manquante.
Si plus d'une donnée manquante
  Message d'erreur pour avertir l'utilisateur.
```

5.5 Fenêtres de dialogue

Dans notre logiciel, nous avons une série de fenêtres de dialogue qui peuvent être réparties en deux groupes :

- Celles qui servent à définir les paramètres du composant que l'on veut placer sur le banc. Ce type de fenêtre n'existe que dans le cas de la lentille, car c'est le seul composant qui nécessite une introduction de paramètres avant le positionnement sur le banc.
- Celles qui servent à visualiser et/ou à modifier les paramètres caractéristiques des différents composants du banc. Ce type de fenêtre existe pour l'objet optique, pour les lentilles, pour l'écran, et pour l'image. Ces fenêtres sont chaque fois structurées de la même façon :
  - Un champ d'édition par paramètre caractéristique.
  - Un bouton "Appliquer" qui applique les modifications afin que l'utilisateur puisse visualiser leurs effets.
  - Un bouton "OK" qui applique les modifications apportées au paramètres et qui ferme la fenêtre de dialogue.
  - Un bouton "Annuler" qui ferme la fenêtre de dialogue et qui restitue au composant ses paramètres de départ.
  - Un bouton "Aide" qui appelle l'aide en ligne, si l'utilisateur veut avoir des informations sur les conventions de notation.

  Remarquons que dans le cas de la fenêtre qui affiche les paramètres de l'image, l'utilisateur ne peut pas faire de modifications.

Les figures 5.1 à 5.5 représentent les cinq différentes fenêtres de dialogue de notre logiciel.

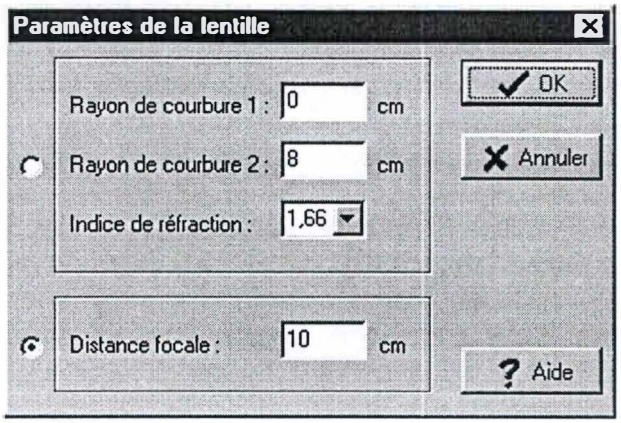

FIG. 5.1: Fenêtre de dialogue pour la définition des paramètres d'une lentille.

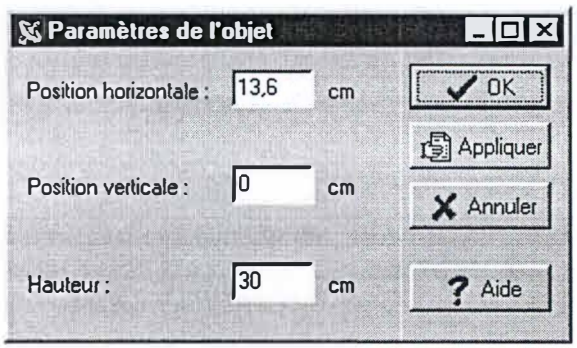

FIG. 5.2: Fenêtre de dialogue pour l'affichage des paramètres de l'objet optique

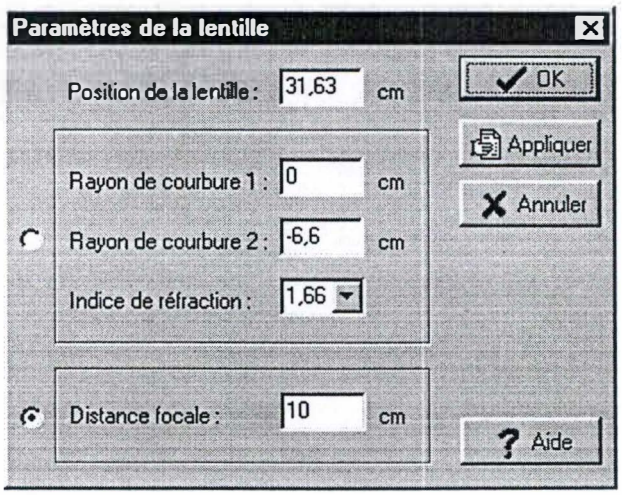

FIG. 5.3: Fenêtre de dialogue pour l'affichage des paramètres d'une lentille

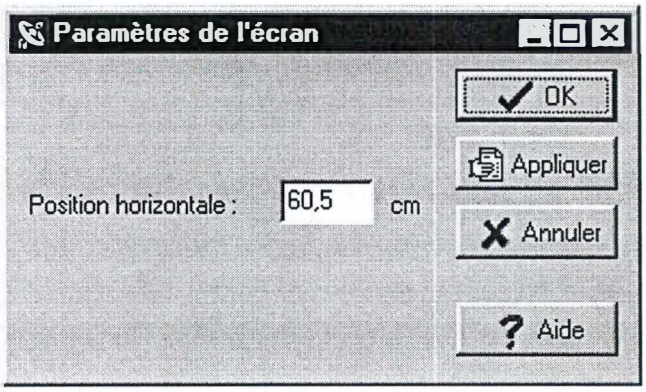

FIG. 5.4: Fenêtre de dialogue pour l'affichage des paramètres de l'écran

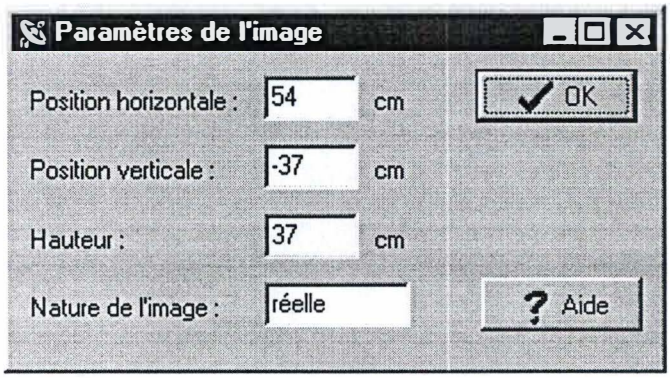

FIG. 5.5: Fenêtre de dialogue pour l'affichage des paramètres de l'image optique

5.6 Objets graphiques relatifs à l'interface de simulation

Concernant la représentation graphique du banc d'optique, nous avions décidé d'afficher une règle graduée au bas de la fenêtre, ainsi qu'une droite horizontale qui sert d'axe optique. Pour obtenir ce résultat, nous avons créé deux objets graphiques (TAxis et TRuler) qui héritent de l'objet TGraphicControl.
Pour la partie "simulation" de notre logiciel nous devons disposer d'une fenêtre dans laquelle on va afficher ce qui est recueilli par l'écran. Dans cette fenêtre nous dessinons l'image recueillie en utilisant notre objet graphique TArrow qui est également un descendant de l'objet TGraphicControl.

5.6.1 Objet "TAxis"

L'objet TAxis est l'objet auquel on fait correspondre l'axe optique du banc. Comme on peut le constater ci-dessous, nous n'avons pas ajouté de nouvelles propriétés à cet objet et les méthodes que nous raffinons ou que nous définissons sont identiques à celles des objets TObjetOpt, TLens et TScreen. Nous ne nous attarderons donc pas sur ce point.

```
TAxis=class(TGraphicControl)
  constructor Create (AOwner : TComponent ; override ;
  procedure Paint ; override ;
  procedure Resize ;
end ;
```

5.6.2 Objet "TRuler"

L'objet TRuler est celui qui va nous servir à représenter la règle graduée de notre banc d'optique.

```
TRuler=class(TGraphicControl)
  Length : real ;
  constructor Create (AOwner : TComponent) ; override ;
  procedure Paint ; override ;
  procedure Resize ;
end ;
```

Une fois de plus, sa structure est très semblable à celles que nous avons rencontrées jusqu'ici. Cependant, il est intéressant de faire deux remarques :

- La propriété Length contient la longueur totale du banc.
- Concernant la méthode Paint, nous dessinons la règle en tenant compte du facteur zoom qui a été sélectionné par l'utilisateur.

5.6.3 Objet "TArrow"

```
TArrow=class(TGraphicControl)
  PosY : integer ;
  Direction : TArrowDirection ;
  Pen : TPen ;
  constructor Create (AOwner : TComponent) ; override ;
  destructor Destroy ; override ;
  procedure Paint ; override ;
  procedure Resize ;
end ;
```

Quelques remarques sur cet objet TArrow :

- La propriété Direction peut prendre une des deux valeurs que nous avons définies dans le type énuméré TArrowDirection : adUp ou adDown.
- La propriété Paint dessine, en fonction de la valeur de Direction, une flèche orientée vers le bas ou vers le haut.

5.6.4 Fenêtre contenant la visualisation de l'écran

La fenêtre où on affiche ce qui est recueilli par l'écran est une simple fenêtre avec l'axe optique au milieu (voir figure 5.6). Nous affichons l'image recueillie en faisant appel à l'objet graphique TArrow décrit plus haut.

Créer la fenêtre

```
Définir les attributs de la fenêtre.
Créer l'objet graphique correspondant à l'axe optique.
Affichage de l'axe optique.
```

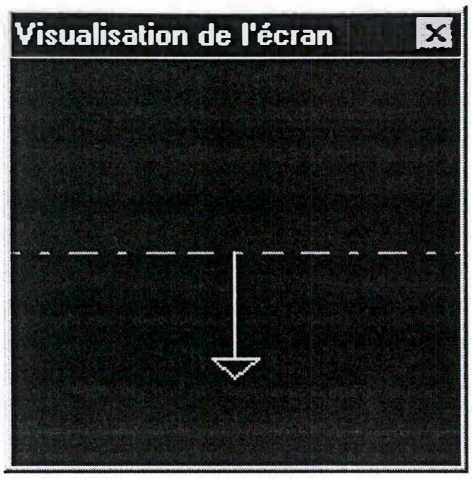

FIG. 5.6: Fenêtre relative à la visualisation de l'écran

Rafraîchir la fenêtre

```
Calculer l'image.
Définir les attributs de l'objet graphique correspondant
à la visualisation.
de l'image (sens et position dans la fenêtre).
Définir le style de trait et la couleur du stylo
de cet objet graphique.
Dessiner l'objet graphique.
```

Fermer la fenêtre

```
Détruire l'objet graphique correspondant à la visualisation
de l'image.
Relâcher le bouton "Visualisation écran".
```

5.7 Coordinateur "Simulation"

5.7.1 Interface relative à la partie "Simulation"

L'interface correspondant à la visualisation de notre banc d'optique est relativement simple. D'une part nous avons la fenêtre qui représente le banc proprement dit, avec la règle graduée et l'axe optique, dans laquelle l'utilisateur va pouvoir placer, déplacer et enlever les composants de son montage; d'autre part nous avons un ensemble de boutons qui permettent à l'utilisateur d'agir sur le banc : choisir le composant à placer, visualiser l'image, ... Normalement, ces boutons devraient être situés dans la fenêtre du banc, mais pour des raisons pratiques, nous avons préféré les placer dans la fenêtre principale du logiciel. Ceci ne devrait pas gêner l'utilisateur parce que les boutons ont été placés juste au-dessus de la fenêtre de simulation (voir Figure 5.7).

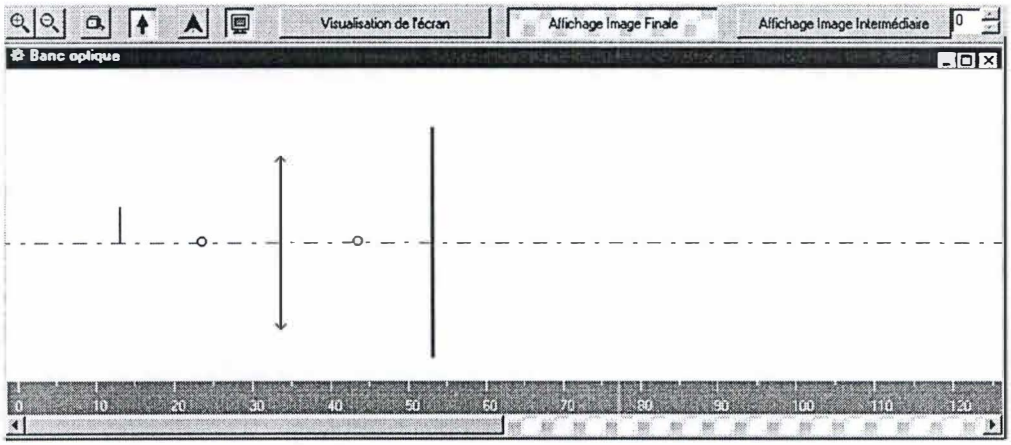

FIG. 5.7: Fenêtre relative à l'aspect "simulation"

5.7.2 Méthodes associées à la fenêtre "Simulation"

Cliquer sur le bouton "Zoom avant"

```
Changer le facteur d'échelle.
Dessiner règle graduée.
Pour tout composant de la liste d'objet
  Dessiner composant.
Si facteur maximal atteint
  Alors désactiver bouton "Zoom avant"
Activer bouton "Zoom arrière".
```

Cliquer sur le bouton "Zoom arrière"

Le déroulement de la méthode est similaire à celui qui a été présenté pour le zoom avant.

Cliquer sur le bouton "Mètre ruban"

```
Désactiver bouton ''Mesure''.
Forme du curseur ← croix.
Style du trait du stylo de la fenêtre ← pointillé.
```

Une fois cette opération faite, on attend que l'utilisateur indique avec la souris le point de départ de la mesure. Pour ce faire, il enfoncera le bouton gauche de la souris ce qui déclenche une méthode décrite plus loin dans le texte.

Cliquer sur le bouton "Objet optique"

```
Si bouton ''Objet Optique'' déjà enfoncé,
  Alors
    Enlever l'objet optique de la liste des composants.
    Détruire l'objet qui correspond à l'objet optique.
    Désactiver les boutons de visualisation de l'image.
  Sinon
    Changer la forme du curseur en flèche verticale.
    Composant à placer ← objet optique.
  Fin si.
Bouton enfoncé ← ¬ bouton enfoncé.
```

Cliquer sur le bouton "Lentille"

Lorsque l'utilisateur enfonce ce bouton, il ne reste pas enfoncé, ce qui laisse la possibilité de placer plusieurs lentilles. Le fait de cliquer sur le bouton ne fait que changer la forme du curseur et signaler le type de composant qu'on souhaite placer.

Cliquer sur le bouton "Ecran"

Le déroulement de la méthode associée au bouton "Ecran" est le même que celui pour le bouton "Objet Optique", car on ne peut également placer qu'un seul écran.

Cliquer sur le bouton "Visualisation de l'écran"

```
Si bouton "Visualisation écran" déjà enfoncé
  Alors
    Fermer fenêtre de visualisation.
    Détruire objet graphique qui représente la flèche.
  Sinon
    Si objet optique dans liste des composants
      Alors
        Créer l'objet graphique qui représente la flèche.
        Afficher fenêtre de visualisation.
      Fin si.
  Fin si.
Bouton enfoncé ← ¬ bouton enfoncé.
```

Cliquer sur le bouton "Affichage image finale"

```
Si bouton "Affichage image finale" déjà enfoncé
  Alors
    Enlever l'image de la liste des composants.
    Détruire objet graphique qui représente l'image.
  Sinon
    Créer objet graphique qui représente l'image.
    Inclure l'image dans la liste des composants.
  Fin si.
Rafraîchir la fenêtre du banc.
Bouton enfoncé ← ¬ bouton enfoncé.
```

Cliquer sur le bouton "Affichage image intermédiaire"

```
Rafraîchir fenêtre du banc.
Bouton enfoncé ← ¬ bouton enfoncé.
```

Créer la fenêtre

```
Créer l'objet graphique représentant l'axe optique.
Créer l'objet graphique représentant la règle graduée.
Définir les attributs de la fenêtre.
```

Redimensionner la fenêtre

```
Redimensionner l'axe optique et la règle graduée.
Pour tout élément de la liste des composants
  Redimensionner le composant.
```

Déplacer souris

```
Afficher positions verticale et horizontale du curseur.
Si forme curseur = croix et bouton gauche enfoncé
  Alors
    Dessiner ligne de la position de départ jusqu'à la
    position actuelle de la souris.
    Afficher distance entre position de départ
    et position actuelle.
  Fin si.
```

Enfoncer bouton de souris

```
Si forme curseur = croix et bouton gauche enfoncé
  Alors retenir position de départ.
Si forme curseur = flèche verticale
  Alors
    Créer l'objet graphique correspondant au composant
    à placer.
    Définir attributs de l'objet.
    Ajouter l'objet à la liste des composants.
    Afficher le composant à l'écran.
    Si composant à placer = objet optique
      Alors activer boutons "Affichage image finale"
      et "Affichage image intermédiaire".
    Si composant à placer = écran
      Alors activer bouton "Visualisation écran".
    Si composant à placer = lentille
      Alors mettre à jour valeur maximale du bouton
      de variation pour l'image intermédiaire.
  Fin si.
```

Relâcher bouton de souris

```
Si forme curseur = croix
  Alors
    Forme du curseur ← défaut.
    Activer bouton "Mesure".
    Style de trait du stylo de la fenêtre ← trait plein.
  Fin si.
```

Déplacer la souris au dessus d'un composant

```
Afficher la position du curseur.
Si forme curseur = croix et bouton gauche enfoncé
  Alors
    Dessiner ligne entre positions initiale
    et actuelle du curseur.
    Afficher distance entre positions initiale
    et actuelle du curseur.
  Fin si.
Si forme curseur = défaut et bouton gauche enfoncé
  Alors position du composant ← position du curseur.
Si forme curseur = défaut et Shift + bouton gauche enfoncés
et composant = objet optique
  Alors sommet objet optique ← position curseur.
Rafraîchir fenêtre de simulation.
```

Enfoncer bouton au dessus d'un composant

```
Si curseur = croix et bouton gauche enfoncé
  Alors retenir position de départ.
Si bouton gauche et Ctrl enfoncés
et composant ≠ image optique
  Alors
    Enlever composant de la liste.
    Détruire objet graphique correspondant au composant.
    Si objet à enlever = objet optique
      Alors désactiver boutons "Affichage image finale"
      et "Affichage image intermédiaire".
    Si objet à enlever = écran
      Alors désactiver bouton "Visualisation de l'écran".
    Si objet à enlever = lentille
      Alors mettre à jour la valeur maximale du bouton
      de variation pour l'image intermédiaire.
    Désactiver les boutons voulus.
    Si composant enlevé = lentille
      Alors mettre à jour la valeur maximale associée au
      bouton de variation utilisé pour sélectionner
      l'image intermédiaire à afficher.
  Fin si.
Rafraîchir la fenêtre de simulation.
```

Relâcher bouton de souris au dessus d'un composant

```
Si curseur = croix
  Alors
    Forme du curseur ← défaut.
    Activer le bouton "Mesure".
    Style de trait du stylo ← trait plein.
  Fin si.
```

Double-cliquer sur lentille

```
Afficher fenêtre de dialogue contenant les paramètres.
```

Le déroulement des méthodes associées au double-cliquer sur les autres composants est identique à celui ci-dessus.

Afficher fenêtre de simulation

```
Si bouton "Affichage image finale" est enfoncé
  Alors calculer image.
Si bouton "Visualisation écran est enfoncé
  Alors rafraîchir fenêtre de visualisation.
```

5.8 Coordinateur "Equations"

5.8.1 Interface relative à la partie "Equations"

La fenêtre qui est associée à la partie "Equations" de notre logiciel est conçue de sorte que l'utilisateur puisse y représenter numériquement un montage optique. Ceci implique naturellement qu'on doit y afficher un grand nombre de données, ce qui explique la structure un peu complexe de cette fenêtre (voir figure 5.8).

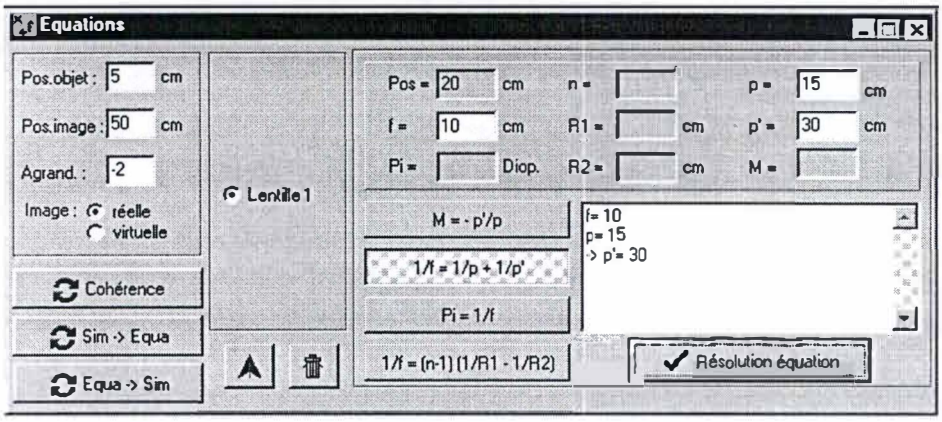

FIG. 5.8: Fenêtre relative à l'aspect "Equations"

Comment l'utilisateur va-t-il représenter un montage ?

- Il dispose d'un champ d'édition pour introduire la position de l'objet optique.
- Il peut ajouter ou retirer une lentille. Cela se fait à l'aide des boutons "Lentille" et "Poubelle". A chaque lentille est associé un bouton radio dans la boîte de regroupement du milieu. En sélectionnant la lentille voulue, il peut introduire ou modifier les valeurs des paramètres grâce aux champs d'édition prévus à cet effet.
- Il est aussi possible d'indiquer la position, la nature et le sens de l'image finale.

5.8.2 Méthodes associées à la fenêtre "Equations"

Cliquer sur le bouton "Vérification cohérence"

```
Appeler la procédure de vérification de la cohérence.
```

Cliquer sur le bouton "Transfert Simulation → Equations"

```
Vérifier la présence d'un objet optique sur le banc.
Si pas image dans la liste des composants
  Alors
    Créer objet graphique correspondant à l'image.
    Insérer image dans la liste des composants.
  Fin si.
Calculer les paramètres de l'image.
Afficher position de l'objet optique dans la fenêtre
"Equations".
Afficher paramètres de l'image dans la fenêtre "Equations".
Pour toute lentille dans la liste des composants
  Faire
    Ajouter lentille dans la fenêtre "Equations".
    Copier paramètres de la lentille.
  Fin faire.
Sélectionner bouton radio de la première lentille.
```

Cliquer sur le bouton "Transfert Equations→ Simulation

```
Vérifier la cohérence des données de la partie "Equations".
Si pas objet optique/image dans la liste des composants
  Alors
    Créer objet graphique correspondant à l'objet
    optique/image.
    Copier paramètres de l'objet graphique/image.
    Ajouter objet optique/image dans la liste des
    composants.
  Fin si.
Pour toute lentille de la partie "Equations"
  Faire
    Créer objet graphique correspondant à une lentille.
    Copier paramètres de la lentille.
    Ajouter objet graphique à la liste des composants.
  Fin faire.
Enfoncer bouton "Affichage image finale".
Rafraîchir fenêtre de simulation.
```

Cliquer sur le bouton "Résolution Equation"

Appeler la procédure de résolution d'équation.

Cliquer sur une des équations

Mettre en évidence les champs d'édition des paramètres intervenant dans l'équation choisie.

Cliquer sur le bouton radio d'une lentille

Afficher dans champs d'édition les paramètres de lentille sélectionnée.

Mettre à zéro les champs d'édition

Mettre à zéro tous les champs d'édition.

Créer la fenêtre "Equations"

Définir les attributs de la fenêtre.

5.9 Coordinateur "Exercices"

5.9.1 Interface relative à la partie "Exercices"

La fenêtre qui contient la partie "Exercices" du logiciel est représentée à la figure 5.9. Elle est constituée d'un champ d'édition multi-linéaire où on affiche l'énoncé de l'exercice, d'une boîte de regroupement garnie d'un certain nombre de champs d'édition où l'utilisateur doit introduire ses réponses, et de trois boutons ("Correction", "Précédent" et "Suivant").

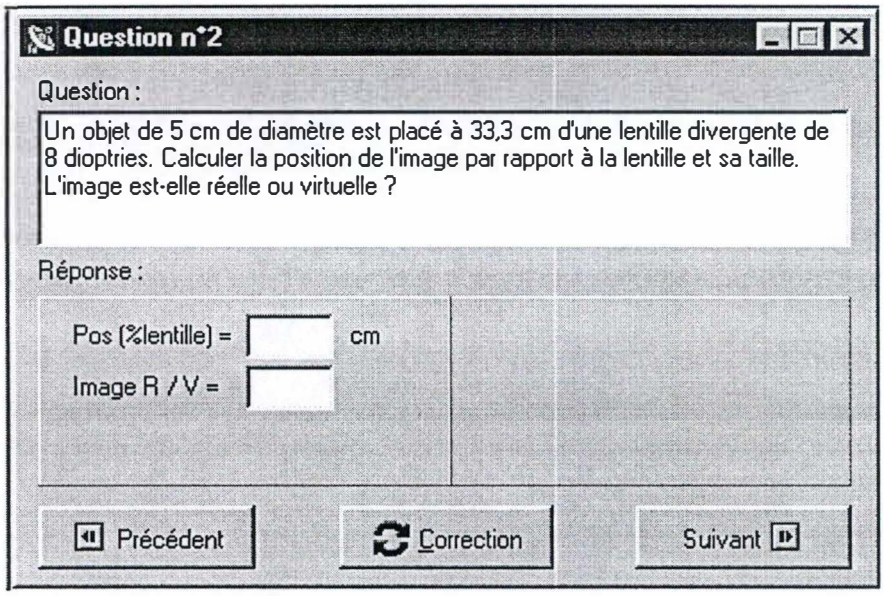

FIG. 5.9: Fenêtre relative à l'aspect "Exercices"

5.9.2 Méthodes associées à la fenêtre "Exercices"

Vu que nous n'avons pas implémenté la base de données qui contient les exercices du logiciel, mais que nous désirions tout de même montrer l'utilisation de cette partie du logiciel, nous avons simplement intégré les exercices dans le code du programme. La méthode implémentée pour la résolution n'est donc pas une méthode générique : elle ne peut pas être appliquée à n'importe quel exercice, mais est spécifique aux exercices donnés.

Cliquer sur le bouton "Suivant"

```
Afficher l'énoncé de l'exercice suivant.
Si dernier exercice
  Alors désactiver bouton "Suivant".
Activer bouton "Précédent".
```

Cliquer sur le bouton "Précédent"

```
Afficher l'énoncé de l'exercice précédent.
Si premier exercice
  Alors désactiver bouton "Précédent".
Activer bouton "Suivant"
```

Cliquer sur "Résolution"

```
Vérifier la présence de l'objet optique sur le banc.
Vérifier l'exactitude du nombre de lentilles.
Vérifier la présence de l'écran à l'endroit où
est formée l'image.
Vérifier l'exactitude des valeurs introduites
dans les champs d'édition.
Envoyer le message d'erreur relatif à l'étape
de la correction.
```

Créer la fenêtre "Exercices"

```
Définir les attributs de la fenêtre.
Afficher l'énoncé du premier exercice.
```

Mettre à zéro la fenêtre "Exercices"

```
Mettre à zéro les différents champs d'édition.
```

5.10 Coordinateur Général

5.10.1 Interface relative à la partie générale

L'interface associée au coordinateur général est formée par la fenêtre principale du logiciel dans laquelle on affiche les trois autres fenêtres (simulation, équations et exercices). Cette fenêtre est également munie de trois boutons : un pour la mise à zéro de l'entièreté du simulateur et deux pour le choix du mode de fonctionnement ("Libre" ou "Problème")(voir figure 5.10).

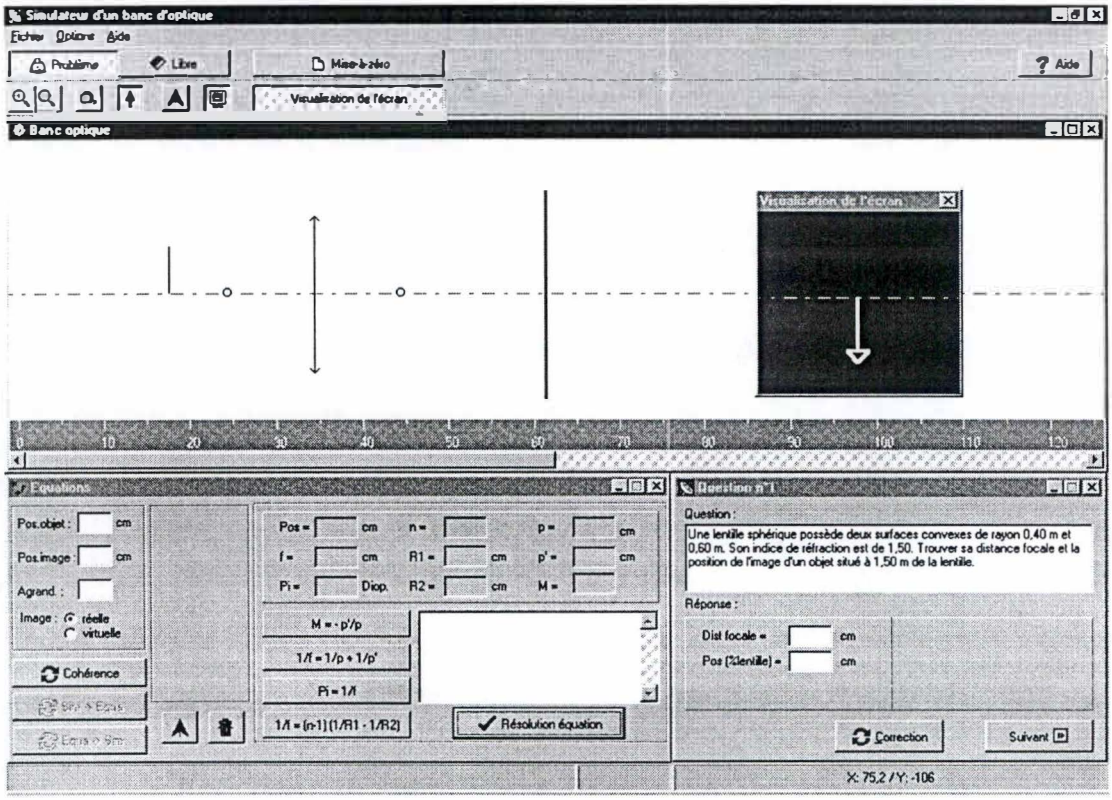

FIG. 5.10: Fenêtre principale du logiciel

5.10.2 Méthodes associées à la fenêtre principale

Cliquer sur le bouton "Mise à zéro"

```
Mettre à zéro la partie ''Equations''.
Mettre à zéro la partie ''Exercices''.
Mettre à zéro la partie ''Simulation''.
```

Cliquer sur le bouton "Libre"

```
Cacher la fenêtre ''Exercices''.
Afficher les boutons ''Affichage image finale''
et '' Affichage image intermédiaire''.
Activer les boutons ''Transfert équations ← simulation''
et ''Transfert simulation ← équations''.
```

Cliquer sur le bouton "Problèmes"

```
Mettre à zéro les différentes parties du logiciel.
Afficher la fenêtre ''Exercices''.
Cacher les boutons ''Affichage image finale''
et ''Affichage image intermédiaire''.
Désactiver les boutons ''Transfert équations ← simulation''
et ''Transfert simulation ← équations''.
```

Conclusion

De nos jour, on entend de plus en plus parler du terme "Enseignement Assisté par Ordinateur" (EAO). Les enseignants, les élèves et les informaticiens réalisent que l'ordinateur peut trouver sa place dans l'enseignement actuel. Malheureusement, il n'est pas facile de créer un "bon" logiciel éducatif : il ne faut pas que l'élève soit un simple exécutant, mais il lui faut au contraire une certaine liberté d'action pour qu'il puisse explorer le monde mis à sa disposition dont il veut approfondir la connaissance. Naturellement, il ne faut pas que cette liberté soit trop grande, auquel cas l'élève risque de perdre le fil de sa recherche. C'est l'étude de cet aspect pédagogique qui a été un des points essentiels dans la réalisation de ce mémoire. Au début de ce travail, nous avions en tête de réaliser uniquement un simulateur "classique", mais au fur et à mesure de nos avancements, nous avons été amené à élaborer un simulateur "intelligent". Ce simulateur permet plus que visualiser les phénomènes : dans notre projet, nous avons combiné le volet pratique et le volet théorique de l'optique géométrique afin d'essayer de mieux faire percevoir ce lien aux élèves du secondaire.

Lors de l'élaboration de notre logiciel, nous avons réalisé qu'il est difficile de cerner et de définir les besoins de l'utilisateur. A tout moment, il est important de jeter le regard de l'utilisateur sur le programme, pour mieux percevoir ses besoins, et mieux évaluer l'utilité de ce qui lui est proposé. Cette démarche n'est pas facile, et a même parfois été négligée dans les premiers temps. Cela a alors naturellement posé problème par la suite, car c'est seulement après l'implémentation et lorsqu'on utilise soi-même le programme qu'on s'aperçoit des perfectionnements à apporter au travail.
Cette prise de conscience constitue le deuxième point essentiel de ce travail.

Ce mémoire nous a également permis de nous familiariser un peu plus avec la programmation orientée-objet (et en particulier avec Delphi). Jusqu'à présent, nous avions déjà utilisé les concepts de base de ce type de programmation, mais ce mémoire nous a donné l'occasion d'explorer et d'approfondir l'aspect graphique

qui nous était moins connu.

En résumé, on peut donc dire que la réalisation de ce mémoire a été enrichissante à trois niveaux :

- *Niveau pédagogique* : recherches réalisées au sujet de l'EAO, et prise de conscience de la difficulté d'élaborer un bon logiciel didactique.
- *Niveau analyse* : importance de cerner le point de vue et les besoins de l'utilisateur.
- *Niveau programmation* : programmation orientée-objet avec son aspect graphique.

Le logiciel ne réagit malheureusement pas toujours comme il le devrait : il y a, par exemple, parfois des problèmes lors de la vérification des données de la partie "Equations" (et donc également lors du transfert de ces données vers le banc). Mais à part la résolution de ces problèmes, quelles sont les extensions possibles que l'on pourrait envisager pour ce logiciel éducatif?

Tout d'abord, il y a la partie "visualisation des aberrations" qui faisait partie de notre projet initial, mais qui n'a finalement pas été implémentée. Pour arriver à cela, il faut encore rajouter tout un ensemble de fonctionnalités. Pour pouvoir visualiser l'aberration chromatique, l'élève devrait pouvoir faire varier la longueur d'onde de la lumière émise par l'objet optique (lumière blanche ou lumière d'une longueur d'onde bien précise). Il faudrait également avoir la possibilité de disposer d'un diaphragme à ouverture circulaire ou annulaire réglable. Le diaphragme à ouverture circulaire permettrait de laisser passer uniquement les rayons centraux d'un faisceau, tandis que le diaphragme à ouverture annulaire ne laisse passer que les rayons marginaux.

A côté de ces projets qui n'ont pas été implémentés, il y a encore d'autres perspectives possibles :

- Plutôt que de travailler avec des lentilles minces, on pourrait envisager de faire des montages avec des lentilles épaisses.
- Il serait intéressant de pouvoir visualiser la trajectoire des rayons lumineux au sein de la lentille.
- Nous avons supposé que le milieu dans lequel l'élève réalise les montages est l'air, comme en laboratoire, mais on pourrait envisager de pouvoir modifier l'indice de réfraction du milieu extérieur.

- On devrait également donner la possibilité à l'élève de défaire ("Undo") et refaire ("Redo") une ou plusieurs actions, pour lui permettre de revenir en arrière.

Bibliographie

[Can96] M. Cantu. *Mastering Delphi 2 for Windows 95/NT.* Sybex, San Francisco, 1996.

[Dub98] E. Dubois. *Méthodologie de Développement de Logiciels (Notes de cours).* FUNDP, 1997–1998.

[Duc91] Ch. Duchâteau. *Trois visages de l'EAO.* OSE Contacts 12, 1991.

[Duc94] Ch. Duchâteau. *Socrate au pays des ordinateurs.* La revue de l'EPI Juin 74, 1994.

[JW76] F.A. Jenkins and H.E. White. *Fundamentals of Optics.* McGraw-Hill, 1976.

[OCD89] OCDE. *Les technologies de l'information et l'éducation. Choisir les bons logiciels.* CERI, OCDE, Paris, 1989.

[Swi] G. Swinnen. La simulation expérimentale : objectifs et contraintes. http ://www.ulg.ac.be/cifen/inforef/swi/simulati.htm.

Annexe A : Listing du programme

unit Arrow;

```
interface

uses Controls, Graphics, Classes, Windows;

type
  TArrowDirection = (adUp, adDown);

  TArrow = class(TGRAPHICCONTROL)
    PosY: integer;
    Pen: TPen;
    Direction: TArrowDirection;
    constructor Create (AOwner: TComponent); override;
    destructor Destroy; override;
    procedure RepaintRequest(Sender: TObject);
    procedure Paint; override;
    procedure Resize;
  end;

implementation

uses Banc, ScrDispl;

constructor TArrow.Create (AOwner: TComponent);
begin
  inherited Create(AOwner);
  Direction:=adUp;
  Pen:=TPen.Create;
  Pen.Color:=clWhite;
  Pen.onChange:=RepaintRequest;
  Left:=round(Form5.ClientWidth/2)-20;
  Width:=40;
  Height:=60;
  Top:=round((Form5.ClientHeight/2)-(Height/2));
end;

destructor TArrow.Destroy;
begin
  Pen.Free;
  inherited Destroy;
end;

procedure TArrow.RepaintRequest(Sender: TObject);
begin
  Invalidate;
end;

procedure TArrow.Paint;
var
  XCenter, YCenter: Integer;
  ArrowPoints: array[0..3] of TPoint;
begin
  inherited Paint;
  { Compute the center }
  YCenter:=(Height-1) div 2;
  XCenter:=(Width-1) div 2;
  { Set the color of the pen }
  Canvas.Pen:=Pen;

  { Draw the line and compute the triangle for the arrow }
  case Direction of
    adUp:
    begin
      Canvas.MoveTo(XCenter,Height);
      Canvas.LineTo(XCenter,round(Height/4+10));
      ArrowPoints[0]:=Point(10,round(Height/4+10));
      ArrowPoints[1]:=Point(XCenter,10);
      ArrowPoints[2]:=Point(Width-11,round(Height/4+10));
      ArrowPoints[3]:=Point(10,round(Height/4+10));
    end;
    adDown:
    begin
      Canvas.MoveTo(XCenter,0);
      Canvas.LineTo(XCenter,Height-round(Height/4));
      ArrowPoints[0]:=Point(XCenter,Height-round(Height/10));
      ArrowPoints[1]:=Point(10,Height-round(Height/4));
      ArrowPoints[2]:=Point(Width-11,Height-round(Height/4));
      ArrowPoints[3]:=Point(XCenter,Height-round(Height/10));
    end;
  end;

  { draw the arrow point }
  Canvas.PolyLine(ArrowPoints);
end;

procedure TArrow.Resize;
begin
  Top:=round(Axe.Top-PosY-Height);
end;

end.
```

unit Axis;

```
interface

uses Controls, Classes, Graphics, Dialogs, Messages;

type
  TAxis=class(TGraphicControl)
```

```
    Pen: TPen;
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Paint; override;
    procedure Resize;
  end;

implementation

uses Banc;

constructor TAxis.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Pen:=TPen.Create;
  Pen.Color:=clRed;
  Pen.Style:=psDashDot;
  Left:=0;
  Height:=1;
  onMouseMove:=Form2.AxisMouseMove;
end;

destructor TAxis.Destroy;
begin
  Pen.Free;
  inherited Destroy;
end;

procedure TAxis.Paint;
begin
  inherited Paint;
  Canvas.Pen:=Pen;
  Canvas.MoveTo(0,0);
  Canvas.LineTo(Width,0);
end;

procedure TAxis.Resize;
begin
  Top:=round(Form2.ClientHeight/2);
  Width:=R.Width;
end;


end.
```

unit Banc;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  ComCtrls, Ruler, Axis, Buttons, ExtCtrls, Screen, StdCtrls,
Arrow,
  Menus, ObjetOpt, Lens;

type
  TForm2 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormMouseMove(Sender: TObject; Shift:
TShiftState; X,
      Y: Integer);
    procedure FormMouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseUp(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure AxisMouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Integer);
    procedure ObjetMouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Integer);
    procedure ObjetMouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure ObjetMouseUp(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure LensDblClick(Sender: TObject);
    procedure ObjOptDblClick(Sender: TObject);
    procedure ImOptDblClick(Sender: TObject);
    procedure ScreenDblClick(Sender: TObject);
   end;


var
  Form2: TForm2;
  R: TRuler;
  Axe: TAxis;
  Ecran: TScreen;
  Fleche: TArrow;
  Objet: TObjetOpt;
  min,max,FirstX, FirstY, LastX: integer;
  scale: real;
  iLensGlob: integer;
```

```
implementation

uses Simul, DLens, ScrDispl, InfoLens, Equation, InfoObjOpt,
Question,
  InfImOpt, InfoScreen, UTypes, ListeObjets, CalculImage;

{$R *.DFM}

{----- Méthodes concernant Form -----}

procedure TForm2.FormCreate(Sender: TObject);
var
  X,Y: Integer;
begin
  ListeComp:=TLISTEComp.Create;
  R:=TRuler.Create(self);
  R.Parent:=self;
  R.Length:=251;
  R.Left:=0;
  Axe:=TAxis.Create(self);
  Axe.Parent:=self;
  min:=0;
  max:=110;
  R.Resize;
  Axe.Resize;
  Scale:=R.Length/(R.Width-30);
  LensNumber:=0;
end;

procedure TForm2.FormResize(Sender: TObject);
var
  i: integer;
begin
  R.Resize;
  Axe.Resize;
  Refresh;
  for i:=0 to ListeComp.count-1 do
    begin
      case TCOMP(ListeComp.items[i]).TypeComp of
        OScreen: TSCREEN(ListeComp.items[i]).Resize;
        OObjetOpt: TObjetOpt(ListeComp.items[i]).Resize;
        OLens: TLENS(ListeComp.Items[i]).Resize;
        OImageOpt: TObjetOpt(ListeComp.items[i]).Resize;
      end;
    end;
  Refresh;
end;

procedure TForm2.FormMouseMove(Sender: TObject; Shift:
TShiftState; X,
  Y: Integer);
var
  Dist, PosX, PosY, Scale: Real;
begin
  Scale:=R.Length/(R.Width-30);
  PosX:=(HorzScrollBar.Position+(X-10))*Scale;
  PosY:=Axe.Top-Y;
  Form1.StatusBar1.Panels[2].Text:='X:
'+FloatToStrF(PosX,ffFixed,15,1)+' / Y:
'+FloatToStrF(PosY,ffFixed,12,0);
  Canvas.MoveTo(FirstX,FirstY);
  if (Cursor=crCross) and (Shift=[ssLeft]) then
    begin
      Canvas.Pen.Mode:=pmXor;
      Canvas.LineTo(LastX,FirstY);
      LastX:=X;
      Canvas.MoveTo(FirstX,FirstY);
      Canvas.LineTo(LastX,FirstY);
      Dist:=(X-FirstX)*scale;

Form1.StatusBar1.Panels[1].Text:=FloatToStrF(Dist,ffFixed,6,1)+
' cm';
    end;
end;

procedure TForm2.FormMouseDown(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  scale: real;
begin
  if Cursor=crCross then
    begin
      FirstX:=X;
      FirstY:=Y;
      LastX:=X;
    end;
  if Cursor=crUpArrow then
    begin
      if CompPut=OScreen then
        begin
          Ecran:=TScreen.Create(self);
          Ecran.Parent:=self;
          Ecran.TypeComp:=OScreen;
          Scale:=R.Length/(R.Width-30);
          Ecran.Top:=round((Form2.ClientHeight/2)-30);
          Ecran.Position:=(HorzScrollBar.Position+(X-
10))*scale;
          Ecran.Left:=X;
          Ecran.Resize;
          Cursor:=crDefault;
          ListeComp.add(Ecran);
          ListeComp.TriComp;
          Form1.SpeedButton2.Enabled:=true;
```

```
        end
      else if CompPut=OObjetOpt then
        begin
          Objet:=TObjetOpt.Create(self);
          Objet.Parent:=self;
          Objet.TypeComp:=OObjetOpt;
          Scale:=R.Length/(R.Width-30);
          Objet.Position:=(HorzScrollBar.Position+(X-
10))*scale;
          Objet.PosY:=0;
          Objet.Top:=round(Axe.Top-Objet.PosY-Objet.Height);
          Objet.Left:=X;
          Objet.Resize;
          Cursor:=crDefault;
          ListeComp.add(Objet);
          Form1.SpeedButton1.Enabled:=true;
        end
      else if CompPut=OLens then
        begin
          Lentille.Height:=round(ClientHeight/2);
          Lentille.Width:=3;
          Scale:=R.Length/(R.Width-30);
          Lentille.Position:=(HorzScrollBar.Position+(X-
10))*scale;
          Lentille.Top:=round(Axe.Top-Lentille.Height/2);
          Lentille.Left:=X;
          Lentille.Resize;
          Cursor:=crDefault;
          ListeComp.add(Lentille);
          ListeComp.TriComp;
          Axe.Resize;
          Form1.UpDown1.Max:=ListeComp.CountLens;
        end;
    end;
end;

procedure TForm2.FormMouseUp(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Cursor=crCross then
    begin
      Form2.Cursor:=crDefault;
      Form1.SpeedMeasure.Enabled:=true;
      Canvas.Pen.Style:=psSolid;
      Canvas.Pen.Mode:=pmCopy;
    Refresh;
    end;

end;

{----- Méthodes concernant Axis -----}

procedure TForm2.AxisMouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Integer);
var
  PosX,PosY,Dist: real;
begin
  Scale:=R.Length/(R.Width-30);
  PosX:=(HorzScrollBar.Position+(X-10))*Scale;
  PosY:=Y;
  Form1.StatusBar1.Panels[2].Text:='X:
'+FloatToStrF(PosX,ffFixed,15,1)+' / Y:
'+FloatToStrF(PosY,ffFixed,12,0);
end;

{----- Méthodes concernant les objets (TCOMP) -----}

procedure TForm2.ObjetMouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Integer);
var
  PosX,PosY,Dist: real;
begin
  Scale:=R.Length/(R.Width-30);
  PosX:=(HorzScrollBar.Position+TCOMP(Sender).Left+(X-
10))*Scale;
  PosY:=Axe.Top-(TCOMP(Sender).Top+Y);
  Form1.StatusBar1.Panels[2].Text:='X:
'+FloatToStrF(PosX,ffFixed,15,1)+' / Y:
'+FloatToStrF(PosY,ffFixed,12,0);
  if (Cursor=crCross) and (Shift=[ssLeft]) then
    begin
      Canvas.MoveTo(FirstX,FirstY);
      Canvas.Pen.Mode:=pmXor;
      Canvas.LineTo(LastX,FirstY);
      LastX:=X+HorzScrollBar.Position+TCOMP(Sender).Left;
      Canvas.MoveTo(FirstX,FirstY);
      Canvas.LineTo(LastX,FirstY);
      Dist:=(X+HorzScrollBar.Position+TCOMP(Sender).Left-
FirstX)*scale;

Form1.StatusBar1.Panels[1].Text:=FloatToStrF(Dist,ffFixed,6,1)+
' cm';
  end;

  if (Cursor=crDefault) and (Shift=[ssLeft]) then
    begin
      case TCOMP(Sender).TypeComp of
         OObjetOpt:
           begin
             TObjetOpt(Sender).PosY:=round(Axe.Top-
(TCOMP(Sender).Top+Y));
             TCOMP(Sender).Top:=round(Axe.Top-
TObjetOpt(Sender).PosY-TCOMP(Sender).Height);
```

```
          end;
      end;

TCOMP(Sender).Position:=(HorzScrollBar.Position+TCOMP(Sender).L
eft+(X-10))*scale;
      case TCOMP(Sender).TypeComp of
         OScreen: begin
                    TSCREEN(Sender).Resize;
                    if Form1.SpeedButton2.Down then
Form5.Paint;
                  end;
         OObjetOpt: begin
                      TObjetOpt(Sender).Resize;
                      if Form1.SpeedButton2.Down then
Form5.Paint;
                    end;
         OLens: begin
                  TLENS(Sender).Resize;
                  if Form1.SpeedButton2.Down then Form5.Paint;
                end;
      end;
      Refresh;
    end;
  if (Cursor=crDefault) and (Shift=[ssLeft,ssShift]) and
(TCOMP(Sender).Top+Y>=R.Height) and (PosY>=0)
                        and (TCOMP(Sender).TypeComp<>OLens)
then
    begin
      if TCOMP(Sender).TypeComp<>OLens then
TCOMP(Sender).Top:=(TCOMP(Sender).Top+Y);
      case TCOMP(Sender).TypeComp of
           OScreen: begin
                      TCOMP(Sender).Height:=2*(Axe.Top-
TCOMP(Sender).Top);
                      if Form1.SpeedButton2.Down then
Form5.Paint;
                    end;
           OObjetOpt: begin
                        TCOMP(Sender).Height:=round(Axe.Top-
TObjetOpt(Sender).PosY-TCOMP(Sender).Top);
                        if Form1.SpeedButton2.Down then
Form5.Paint;
                      end;
      end;
      Refresh;
    end;
end;

procedure TForm2.ObjetMouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  i: integer;
begin
  if Cursor=crCross then
    begin
      FirstX:=X+HorzScrollBar.Position+TCOMP(Sender).Left;
      FirstY:=TGraphicControl(Sender).Top+Y;
      LastX:=X+HorzScrollBar.Position+TCOMP(Sender).Left;
      Canvas.MoveTo(FirstX,FirstY);
    end;
  if (Shift=[ssLeft,ssCtrl]) and
(TCOMP(Sender).TypeComp<>OImageOpt) then
    begin
      case TCOMP(Sender).TypeComp of
        OScreen: begin
                   Form1.SpeedScreen.Down:=false;
                   Form1.SpeedButton2.Down:=false;
                   Form1.SpeedButton2.Enabled:=false;
                   Form5.Visible:=false;
                   Form1.SpeedScreen.Hint:='Positionner un
écran sur le banc';
                 end;
        OObjetOpt: begin
                     Form1.SpeedArrow.Hint:='Positionner un
objet (flèche) sur le banc';
                     Form1.SpeedArrow.Down:=false;
                     if Form1.SpeedButton1.Down then
                       begin
                         Form1.SpeedButton1.Down:=false;
                         Form1.SpeedButton1.Enabled:=false;
                         Form1.SpeedButton3.Down:=false;
                         Form1.SpeedButton3.Enabled:=false;
                         Image.Destroy;
                         i:=ListeComp.FindTypeComp(OImageOpt);
                         ListeComp.Delete(i);
                         Form2.Paint;
                       end;
                   end;
      end;
      TCOMP(Sender).Destroy;

i:=ListeComp.FindComp(TCOMP(Sender).Position,TCOMP(Sender).Type
Comp);
      ListeComp.Delete(i);
      if strtoint(Form1.Edit1.Text)>ListeComp.CountLens then
Form1.Edit1.Text:=inttostr(ListeComp.CountLens);
      Form1.UpDown1.Max:=ListeComp.CountLens;
      if ListeComp.CountLens=0 then
        begin
          Form1.SpeedButton1.Down:=false;
          Form1.SpeedButton2.Down:=false;
          Form1.SpeedButton3.Down:=false;
          if ListeComp.FindTypeComp(OImageOpt)<>-1 then
            begin
```

```
            Image.Destroy;
            i:=ListeComp.FindTypeComp(OImageOpt);
            ListeComp.Delete(i);
          end;
      end;
    Refresh;
  end;
end;

procedure TForm2.ObjetMouseUp(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Cursor=crCross then
    begin
      Form2.Cursor:=crDefault;
      Form1.SpeedMeasure.Enabled:=true;
      Canvas.Pen.Style:=psSolid;
      Canvas.Pen.Mode:=pmCopy;
      Refresh;
    end;
end;

procedure TForm2.LensDblClick(Sender: TObject);
var
  Q1,Q2: Real;
  OldPos, OldRadius1, OldRadius2, OldFocus1, OldRefrInd: Real;
begin
  OldPos:=TLens(Sender).Position;
  OldRadius1:=TLens(Sender).Radius1;
  OldRadius2:=TLens(Sender).Radius2;
  OldFocus1:=TLens(Sender).Focus1;
  OldRefrInd:=TLens(Sender).RefrInd;

Form6.Edit1.Text:=floattostrF(TLENS(Sender).Radius1,ffGeneral,4
,1);

Form6.Edit2.Text:=floattostrF(TLENS(Sender).Radius2,ffGeneral,4
,1);

Form6.Edit3.Text:=floattostrF(TLENS(Sender).Focus1,ffGeneral,4,
1);

Form6.Edit4.Text:=floattostrF(TLENS(Sender).Position,ffGeneral,
4,1);

Form6.ComboBox1.Text:=floattostrF(TLENS(Sender).RefrInd,ffGener
al,4,1);
  ListeComp.TriComp;
  iLensGlob:=ListeComp.FindComp(TLens(Sender).Position, OLens);
  if Form6.ShowModal=mrOk then
    begin
      TLens(Sender).Position:=strtofloat(Form6.Edit4.Text);
      if Form6.RadioButton1.Checked then
        begin
          TLens(Sender).Radius1:=strtofloat(Form6.Edit1.Text);
          TLens(Sender).Radius2:=strtofloat(Form6.Edit2.Text);

TLens(Sender).RefrInd:=strtofloat(Form6.ComboBox1.Text);
          if TLens(Sender).Radius1=0 then Q1:=0 else
Q1:=1/TLens(Sender).Radius1;
          if TLens(Sender).Radius2=0 then Q2:=0 else
Q2:=1/TLens(Sender).Radius2;
          TLens(Sender).Focus1:=1/((TLens(Sender).RefrInd-
1)*(Q1-Q2));
        end
      else if Form6.RadioButton2.Checked then
        begin
          TLens(Sender).Focus1:=strtofloat(Form6.Edit3.Text);
          TLens(Sender).RefrInd:=1.66;
          TLens(Sender).Radius1:=0;
          TLens(Sender).Radius2:=-
TLens(Sender).Focus1*(TLens(Sender).RefrInd-1);
        end;
    end
  else
    begin
      TLens(Sender).Position:=OldPos;
      TLens(Sender).Radius1:=OldRadius1;
      TLens(Sender).Radius2:=OldRadius2;
      TLens(Sender).RefrInd:=OldRefrInd;
      TLens(Sender).Focus1:=OldFocus1;
    end;
  Refresh;
end;

procedure TForm2.ObjOptDblClick(Sender: TObject);
begin
  Form7.showmodal;
end;

procedure TForm2.ImOptDblClick(Sender: TObject);
begin
  Form9.showmodal;
end;

procedure TForm2.ScreenDblClick(Sender: TObject);
begin
  Form10.Showmodal;
end;

procedure TForm2.FormPaint(Sender: TObject);
begin
  if Form1.SpeedButton1.Down then CalculIm;
  if Form1.SpeedButton2.Down then Form5.Paint;
```

```
end;

end.
```

unit CalculEqua;

```
interface

uses SysUtils, Dialogs;

function VerifCoher:boolean;

function Diff(x,y:real):boolean;

implementation

uses Equation,EquaLens,Banc;

function VerifCoher:boolean;
var
  max,i: integer;
  ErrorMsg: string;

PosImTot,Temp,Focus,Position,Radius1,Radius2,Power,RefrInd,Q1,Q
2,PosObj,PosIm,Agr,AgrTot: real;
  VerIm,Def, CoherIm: boolean;
begin
  ErrorMsg:='';
  CoherIm:=true;
  Def:=true;
  VerIm:=true;
  AgrTot:=1;
  for i:=0 to LensList.count-1 do
    begin
      if (TEquaLens(LensList.items[i]).Focus1<>'') and
(TEquaLens(LensList.items[i]).Power<>'') then
        begin

Focus:=strtofloat(TEquaLens(LensList.items[i]).Focus1);

Power:=strtofloat(TEquaLens(LensList.items[i]).Power);
          if Diff(Focus/100,1/Power)
            then ErrorMsg:=ErrorMsg+'Lentille
'+inttostr(i+1)+': Incohérence entre distance focale et
puissance.'+chr(13)+chr(10);
        end;
      if (TEquaLens(LensList.items[i]).Focus1<>'')
          and ((TEquaLens(LensList.items[i]).RefrInd<>'')
                and (TEquaLens(LensList.items[i]).Radius1<>'')
                and (TEquaLens(LensList.items[i]).Radius2<>''))
then
        begin

Focus:=strtofloat(TEquaLens(LensList.items[i]).Focus1);

Radius1:=strtofloat(TEquaLens(LensList.items[i]).Radius1);
          if Radius1=0 then Q1:=0 else Q1:=1/Radius1;

Radius2:=strtofloat(TEquaLens(LensList.items[i]).Radius2);
          if Radius2=0 then Q2:=0 else Q2:=1/Radius2;

RefrInd:=strtofloat(TEquaLens(LensList.items[i]).RefrInd);
          if Diff(Focus,1/((RefrInd-1)*(Q1-Q2)))
            then ErrorMsg:=ErrorMsg+'Lentille
'+inttostr(i+1)+': Incohérence entre distance focale et
(R1,R2,n).'+chr(13)+chr(10);
        end;
      if (TEquaLens(LensList.items[i]).Power<>'')
          and ((TEquaLens(LensList.items[i]).RefrInd<>'')
                and (TEquaLens(LensList.items[i]).Radius1<>'')
                and (TEquaLens(LensList.items[i]).Radius2<>''))
then
        begin

Focus:=100/strtofloat(TEquaLens(LensList.items[i]).Power);

Radius1:=strtofloat(TEquaLens(LensList.items[i]).Radius1);
          if Radius1=0 then Q1:=0 else Q1:=1/Radius1;

Radius2:=strtofloat(TEquaLens(LensList.items[i]).Radius2);
          if Radius2=0 then Q2:=0 else Q2:=1/Radius2;

RefrInd:=strtofloat(TEquaLens(LensList.items[i]).RefrInd);
          if Diff(Focus,1/((RefrInd-1)*(Q1-Q2)))
            then ErrorMsg:=ErrorMsg+'Lentille
'+inttostr(i+1)+': Incohérence entre puissance et
(R1,R2,n).'+chr(13)+chr(10);
        end;
    end;
  if ErrorMsg<>''
    then MessageDlg(ErrorMsg,mtError,[mbOK],0)
    else
      begin
        for i:=0 to LensList.count-1 do
          begin
            if TEquaLens(LensList.items[i]).Position='' then
Def:=false;
            if (TEquaLens(LensList.items[i]).Focus1='')
              and (TEquaLens(LensList.items[i]).Power='')
              and ((TEquaLens(LensList.items[i]).Radius1='')
                  and (TEquaLens(LensList.items[i]).Radius2='')
```

```
                  and
(TEquaLens(LensList.items[i]).RefrInd='')) then Def:=false;
          end;
        if not(Def)
          then MessageDlg('Votre système n''est pas optiquement
défini:' +chr(13)+chr(10)+
                              'il manque la position d''une
lentille ou'+chr(13)+chr(10)+
                              'une lentille n''a pas été
suffisamment caractérisée.',mtError,[mbOK],0)
          else
            begin
            for i:=0 to LensList.count-1 do
            begin
              if TEquaLens(LensList.items[i]).PosObj='' then
VerIm:=false;
              if TEquaLens(LensList.items[i]).PosIm='' then
VerIm:=false;
            end;
            for i:=0 to LensList.count-1 do
            begin

Position:=strtofloat(TEquaLens(LensList.items[i]).Position);
              if TEquaLens(LensList.items[i]).Focus1<>''
                then
Focus:=strtofloat(TEquaLens(LensList.items[i]).Focus1)
                else if TEquaLens(LensList.items[i]).Power<>''
                  then
Focus:=100/strtofloat(TEquaLens(LensList.items[i]).Focus1)
                    else
                      begin

Radius1:=strtofloat(TEquaLens(LensList.items[i]).Radius1);
                        if Radius1=0 then Q1:=0 else
Q1:=1/Radius1;

Radius2:=strtofloat(TEquaLens(LensList.items[i]).Radius2);
                        if Radius2=0 then Q2:=0 else
Q2:=1/Radius2;

RefrInd:=strtofloat(TEquaLens(LensList.items[i]).RefrInd);
                        Focus:=1/((RefrInd-1)*(Q1-Q2));
                      end;
              if i=0 then PosObj:=Position-
strtofloat(Form3.Edit10.Text)
                else PosObj:=Position-
strtofloat(TEquaLens(LensList.items[i-1]).Position)-
strtofloat(TEquaLens(LensList.items[i-1]).PosIm);
              if Verim then
              begin

temp:=strtofloat(TEquaLens(LensList.items[i]).PosObj);
                if Diff(PosObj,temp) then CoherIm:=false;
                end;
                PosIm:=1/((1/Focus)-(1/PosObj));
                if VerIm then
                begin

temp:=strtofloat(TEquaLens(LensList.items[i]).PosIm);
                if Diff(PosIm,temp) then CoherIm:=false;
                end;
                Agr:=-PosIm/PosObj;
                PosImTot:=Position+PosIm;
                AgrTot:=AgrTot*Agr;
                if VerIm then
                if TEquaLens(LensList.items[i]).Agr<>'' then
                  begin

temp:=strtofloat(TEquaLens(LensList.items[i]).Agr);
                    Agr:=-PosIm/PosObj;
                    if Diff(Agr,temp) then CoherIm:=false;
                  end;
              end;
              if Form3.Edit12.Text<>'' then
                  if Diff(strtofloat(Form3.Edit12.Text),AgrTot)
then CoherIm:=false;
              if Form3.Edit11.Text<>'' then
                begin
                if Diff(strtofloat(Form3.Edit11.Text),PosImTot)
then CoherIm:=false;
                end;
              end;
          if not(CoherIm) then MessageDlg('Erreur de cohérence
au niveau des p et p'' ou au niveau des
agrandissements',mtError,[mbOK],0);
      end;
  if (CoherIm) and (Def) and (ErrorMsg='') then Result:=true
else Result:=false;
end;

function Diff(x,y:real):boolean;
begin
  Result:=false;
  if (x<y-abs(3*y)/100) or (x>y+abs(3*y)/100) then
Result:=true;
end;

end.
```

```
unit CalculImage;

interface

uses SysUtils;

procedure CalculIm;

implementation

uses Simul, Banc, Lens;

procedure CalculIm;
var
  Q1, Q2, R1, R2, n, PosObj, Agr, PosIm, f, f2: double;
  i,j,k: integer;
  numLens,LensPos,last: integer;
  c1,c2,c3,c4,c5,k1,k2,k3,k4,k5: double;

ObjetTempPosition,ObjetTempTop,ObjetTempHeight,ObjetTempPosY,Im
ageTempPosition,ImageTempTop,ImageTempHeight,ImageTempPosY,Imag
eTempLeft: double;
  ImageTempPoint,ObjetTempPoint: double;
  ImageTempReal,ObjetTempUp,ImageTempUp:boolean;
begin
if ListeComp.count>1 then
begin
  ListeComp.TriComp;
  ImageTempTop:=Objet.Top;
  ImageTempPosition:=Objet.Position;
  ImageTempHeight:=Objet.Height;
  ImageTempPosY:=Objet.PosY;
  ImageTempPoint:=Objet.PosY+Objet.Height;
  ImageTempReal:=true;
  ImageTempUp:=true;
  LensPos:=ListeComp.FindNextLens(0);
  numLens:=ListeComp.CountLens;
  if numLens>0 then
  begin
  for i:=0 to numLens-1 do
    begin
      ObjetTempTop:=ImageTempTop;
      ObjetTempPosition:=ImageTempPosition;
      ObjetTempHeight:=ImageTempHeight;
      ObjetTempPosY:=ImageTempPosY;
      ObjetTempPoint:=ImageTempPoint;
      ObjetTempUp:=ImageTempUp;
      c1:=(-TLens(ListeComp.Items[LensPos]).Focus1)/scale;
      if ObjetTempTop<=Axe.Top then c2:=ObjetTempHeight{-0}
else c2:=-(ObjetTempTop-Axe.Top+ObjetTempHeight);
      c3:=c1*c2;
      c4:=c2*TLens(ListeComp.Items[LensPos]).Position/scale;
      c5:=c4-c3;
      k1:=(ObjetTempPosition-
TLens(ListeComp.Items[LensPos]).Position)/scale;
      k2:=c2;
      k3:=k1*c2;
      k4:=k2*ObjetTempPosition/scale;
      k5:=k4-k3;
      if (c2*k1-c1*k2)<>0 then ImageTempPosition:=(c5*k1-
c1*k5)*scale/(c2*k1-c1*k2);
      ImageTempLeft:=round(ImageTempPosition/scale+10-
Form2.HorzScrollBar.Position);
      if (TLens(ListeComp.Items[LensPos]).Position-
ObjetTempPosition)<>0 then Agr:=(ImageTempPosition-
TLens(ListeComp.Items[LensPos]).Position)/(ObjetTempPosition-
TLens(ListeComp.Items[LensPos]).Position)
         else Agr:=1;
      ImageTempHeight:=round(abs(ObjetTempHeight*Agr));

TLens(ListeComp.Items[LensPos]).PosObj:=ObjetTempPosition;
      TLens(ListeComp.Items[LensPos]).PosIm:=ImageTempPosition;
      TLens(ListeComp.Items[LensPos]).Agr:=Agr;
      if Agr<=0 then
        begin
          ImageTempTop:=round(Axe.Top+ObjetTempPosY*abs(Agr));
          ImageTempPosY:=-
(ObjetTempPosY*abs(Agr)+ImageTempHeight);
          if ImageTempUp then ImageTempPoint:=-
(ImageTempHeight+ObjetTempPosY*abs(Agr))
             else
ImageTempPoint:=ImageTempPosY+ImageTempHeight;
          if
ObjetTempPosition>=TLens(ListeComp.Items[LensPos]).Position
then ImageTempReal:=false else ImageTempReal:=true;
          ImageTempUp:=not(ObjetTempUp);
        end
      else
        begin
          ImageTempPosY:=ObjetTempPosY*Agr;
          ImageTempTop:=Axe.Top-ImageTempPosY-ImageTempHeight;
          if ImageTempUp then
ImageTempPoint:=ImageTempPosY+ImageTempHeight
             else ImageTempPoint:=ImageTempPosY;
          if
ObjetTempPosition<=TLens(ListeComp.Items[LensPos]).Position
then ImageTempReal:=false else ImageTempReal:=true;
          ImageTempUp:=ObjetTempUp;
        end;
      if (Form1.SpeedButton3.Down) and
(strtoint(Form1.Edit1.Text)=i+1) then
        begin
          Form2.Canvas.Pen.Color:=clSilver;
          Form2.Canvas.Pen.Style:=psSolid;
```

```
          if i>0 then
            begin

Form2.Canvas.MoveTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ObjetTempPosY));

Form2.Canvas.LineTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-
round(ObjetTempPosY+ObjetTempHeight));
            end;
          if i<numLens-1 then
            begin

Form2.Canvas.MoveTo(round(ImageTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ImageTempPosY));

Form2.Canvas.LineTo(round(ImageTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-
round(ImageTempPosY+ImageTempHeight));
            end;
          Form2.Canvas.Pen.Style:=psDash;

Form2.Canvas.MoveTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ObjetTempPoint));
          if
ObjetTempPosition<=TLens(ListeComp.Items[LensPos]).Position
then
            begin

Form2.Canvas.LineTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top-
round(ObjetTempPoint));
              if ImageTempReal then
                begin

Form2.Canvas.LineTo(round(ImageTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ImageTempPoint));

Form2.Canvas.LineTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ObjetTempPoint));
                end
              else
                begin
                  if TLens(ListeComp.Items[LensPos]).Focus1>=0
then
                    begin
                      Form2.Canvas.Pen.Style:=psDashDotDot;

Form2.Canvas.LineTo(round(ImageTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ImageTempPoint));
                      Form2.Canvas.Pen.Style:=psDash;

Form2.Canvas.LineTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top);

Form2.Canvas.MoveTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top-
round(ObjetTempPoint));

Form2.Canvas.LineTo(round((TLens(ListeComp.Items[LensPos]).Posi
tion+TLens(ListeComp.Items[LensPos]).Focus1)/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top);
                    end
                  else
                    begin
                      Form2.Canvas.Pen.Style:=psDashDotDot;

Form2.Canvas.LineTo(round((TLens(ListeComp.Items[LensPos]).Posi
tion+TLens(ListeComp.Items[LensPos]).Focus1)/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top);
                      Form2.Canvas.Pen.Style:=psDash;

Form2.Canvas.MoveTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ObjetTempPoint));

Form2.Canvas.LineTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top);
                    end;
                end;
            end
          else
            begin
              Form2.Canvas.Pen.Style:=psDashDotDot;

Form2.Canvas.LineTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top-
round(ObjetTempPoint));
              if not ImageTempReal then
                begin

Form2.Canvas.LineTo(round(ImageTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ImageTempPoint));
                  Form2.Canvas.Pen.Style:=psDash;

Form2.Canvas.LineTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ObjetTempPoint));
                end
              else
                begin
                  Form2.Canvas.Pen.Style:=psDash;
                    if
TLens(ListeComp.Items[LensPos]).Focus1>=0 then
                      begin
```

```
Form2.Canvas.LineTo(round((TLens(ListeComp.Items[LensPos]).Posi
tion+TLens(ListeComp.Items[LensPos]).Focus1)/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top);

Form2.Canvas.MoveTo(round(ObjetTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ObjetTempPoint));

Form2.Canvas.LineTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top);
                    end
                  else
                    begin

Form2.Canvas.LineTo(round(ImageTempPosition/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top-round(ImageTempPoint));

Form2.Canvas.LineTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top);
                      Form2.Canvas.Pen.Style:=psDashDotDot;

Form2.Canvas.MoveTo(round(TLens(ListeComp.Items[LensPos]).Posit
ion/scale)+10-Form2.HorzScrollBar.Position,Axe.Top-
round(ObjetTempPoint));

Form2.Canvas.LineTo(round((TLens(ListeComp.Items[LensPos]).Posi
tion+TLens(ListeComp.Items[LensPos]).Focus1)/scale)+10-
Form2.HorzScrollBar.Position,Axe.Top);
                      Form2.Canvas.Pen.Style:=psDash;
                    end;
              end;
          end;
      end;
      last:=LensPos;
      LensPos:=ListeComp.FindNextLens(last+1);
    end;
  end;
  if numLens>0 then
    begin
      Image.Position:=ImageTempPosition;
      if (ImageTempHeight>(Form2.ClientHeight/2)) and (not
ImageTempUp) then Image.Height:=round(Form2.ClientHeight/2)
         else Image.Height:=round(ImageTempHeight);
      if Image.Position>=R.Length then
Image.Left:=round(R.Length/scale-
Form2.HorzScrollBar.Position+10)
         else Image.Left:=round(ImageTempLeft);
      Image.PosY:=round(ImageTempPosY);
      Image.Top:=round(ImageTempTop);
      Image.Point:=ImageTempPoint;
      Image.Real:=ImageTempReal;
      Image.Up:=ImageTempUp;
      Image.Pen.Color:=clBlue;
      if ImageTempReal then Image.Pen.Style:=psSolid else
Image.Pen.Style:=psDot;
    end
  else
    begin
    Image.Pen.Style:=psSolid;
    Image.Pen.Color:=clBlack;
    end;
end;
  Form2.Canvas.Pen.Color:=clBlack;
  Form2.Canvas.Pen.Style:=psSolid;
end;


end.
```

unit DLens;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons, Lens;

type
  TForm4 = class(TForm)
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Edit1: TEdit;
    Label5: TLabel;
    Label2: TLabel;
    Edit2: TEdit;
    Label6: TLabel;
    Label3: TLabel;
    ComboBox1: TComboBox;
    RadioButton1: TRadioButton;
    GroupBox2: TGroupBox;
    Label4: TLabel;
    Edit3: TEdit;
    Label7: TLabel;
    RadioButton2: TRadioButton;
    BitBtn3: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
```

```
    procedure Edit1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form4: TForm4;
  Lentille: TLens;

implementation

uses Banc, Simul, UTypes;

{$R *.DFM}

procedure TForm4.BitBtn1Click(Sender: TObject);
var
  Q1,Q2: Real;
begin
  if (StrToFloat(Edit1.Text)=0) and (StrToFloat(Edit2.Text)=0)
then
    begin
      Showmessage('Rayon 1 et Rayon 2 ne peuvent pas être tous
les deux égaux à 0.');
    end
  else
    begin
      Form2.Cursor:=crUpArrow;
      CompPut:=OLens;
      Lentille:=TLens.Create(Form2);
      if RadioButton1.Checked then
        begin
          with Lentille do
            begin
              Parent:=Form2;
              Height:=0;
              Radius1:=StrToFloat(Edit1.Text);
              Radius2:=StrToFloat(Edit2.Text);
              RefrInd:=StrToFloat(ComboBox1.Text);
              if Radius1=0 then Q1:=0 else Q1:=1/Radius1;
              if Radius2=0 then Q2:=0 else Q2:=1/Radius2;
              Focus1:=1/((RefrInd-1)*(Q1-Q2));
            end;
        end
      else if RadioButton2.Checked then
        begin
          with Lentille do
            begin
              Parent:=Form2;
              Height:=0;
              Radius1:=0;
              RefrInd:=1.66;
              Focus1:=strtofloat(Edit3.Text);
              Radius2:=-Focus1*(RefrInd-1);
            end;
        end;
      Visible:=false;
    end;
end;

procedure TForm4.Edit1Change(Sender: TObject);
begin
  RadioButton1.Checked:=true;
end;

procedure TForm4.Edit3Change(Sender: TObject);
begin
RadioButton2.Checked:=true;
end;

procedure TForm4.Edit1Click(Sender: TObject);
begin
  TEDIT(Sender).SelectAll;
end;

end.
```

unit EquaLens;

```
interface

 uses UTypes, Graphics, Classes, Dialogs, SysUtils;

type
  TEquaLens=class
     LNumber: integer;
     Position: string;
     Radius1: string;
     Radius2: string;
     Focus1: string;
     Power: string;
     RefrInd: string;
     PosObj: string;
     PosIm: string;
     Agr: string;
  end;

implementation
```

```
end.
```

unit Equation;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  Buttons, StdCtrls, ExtCtrls, UTypes, ListeObjets;

type
  TForm3 = class(TForm)
    RadioGroup1: TRadioGroup;
    GroupBox1: TGroupBox;
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label7: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Memo1: TMemo;
    BitBtn1: TBitBtn;
    Label8: TLabel;
    Edit5: TEdit;
    Edit6: TEdit;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Edit7: TEdit;
    Label14: TLabel;
    Edit8: TEdit;
    Label15: TLabel;
    Label16: TLabel;
    Edit9: TEdit;
    Label18: TLabel;
    Label19: TLabel;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpeedButton6: TSpeedButton;
    SpeedButton7: TSpeedButton;
    GroupBox2: TGroupBox;
    Label6: TLabel;
    Edit10: TEdit;
    Label17: TLabel;
    Edit11: TEdit;
    Label13: TLabel;
    Label20: TLabel;
    Label21: TLabel;
    Label9: TLabel;
    Edit12: TEdit;
    BitCoher: TBitBtn;
    Label23: TLabel;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    BitSimEqua: TBitBtn;
    BitEquaSim: TBitBtn;
    procedure SpeedButton1Click(Sender: TObject);
    procedure EditReset;
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton7Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure BitCoherClick(Sender: TObject);
    procedure EditTextReset;
    procedure Edit10Click(Sender: TObject);
    procedure BitSimEquaClick(Sender: TObject);
    procedure BitEquaSimClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  function NoBlankValues: boolean;


var
  Form3: TForm3;
  LensNumber: integer;
  LensList: TLensList;

implementation

{$R *.DFM}

uses EquaLens, ObjetOpt, Banc, Lens, Simul, QuestList,
CalculImage, CalculEqua;
```

```
procedure TForm3.EditReset;
begin
  Edit1.Color:=clMenu;
  Edit2.Color:=clMenu;
  Edit3.Color:=clMenu;
  Edit4.Color:=clMenu;
  Edit5.Color:=clMenu;
  Edit6.Color:=clMenu;
  Edit7.Color:=clMenu;
  Edit8.Color:=clMenu;
  Edit9.Color:=clMenu;
end;

procedure TForm3.SpeedButton1Click(Sender: TObject);
begin
   if SpeedButton1.Down then
     begin
       EditReset;
       Edit7.Color:=clWindow;
       Edit8.Color:=clWindow;
       Edit9.Color:=clWindow;
     end
end;

procedure TForm3.SpeedButton2Click(Sender: TObject);
begin
   if SpeedButton2.Down then
     begin
       EditReset;
       Edit2.Color:=clWindow;
       Edit7.Color:=clWindow;
       Edit8.Color:=clWindow;
     end
end;

procedure TForm3.SpeedButton3Click(Sender: TObject);
begin
   if SpeedButton3.Down then
     begin
       EditReset;
       Edit2.Color:=clWindow;
       Edit3.Color:=clWindow;
     end
end;

procedure TForm3.SpeedButton4Click(Sender: TObject);
begin
   if SpeedButton4.Down then
     begin
       EditReset;
       Edit2.Color:=clWindow;
       Edit4.Color:=clWindow;
       Edit5.Color:=clWindow;
       Edit6.Color:=clWindow;
     end
end;

procedure TForm3.BitBtn1Click(Sender: TObject);
var
  Res: string;
  Q1,Q2: Real;
begin
  if SpeedButton1.Down then
    begin
      if (Edit9.Text='') and (Edit8.Text<>'') and
(Edit7.Text<>'') then
        begin
          Memo1.Lines.add('p= '+(Edit7.Text));
          Memo1.Lines.add('p''= '+(Edit8.Text));
          Res:=floattostrF(-
strtofloat(Edit8.Text)/strtofloat(Edit7.Text),ffGeneral,3,1);
          Memo1.Lines.add('-> M= '+Res);
          Memo1.Lines.add('');
          Edit9.Text:=Res;
        end
      else if (Edit8.Text='') and (Edit9.Text<>'') and
(Edit7.Text<>'') then
        begin
          Memo1.Lines.add('p= '+(Edit7.Text));
          Memo1.Lines.add('M= '+(Edit9.Text));
          Res:=floattostrF(-
strtofloat(Edit9.Text)*strtofloat(Edit7.Text),ffGeneral,3,1);
          Memo1.Lines.add('-> p''= '+Res);
          Memo1.Lines.add('');
          Edit8.Text:=Res;
        end
      else if (Edit7.Text='') and (Edit9.Text<>'') and
(Edit8.Text<>'') then
        begin
          Memo1.Lines.add('p''= '+(Edit8.Text));
          Memo1.Lines.add('M= '+(Edit9.Text));
          Res:=floattostrF(-
strtofloat(Edit8.Text)/strtofloat(Edit9.Text),ffGeneral,3,1);
          Memo1.Lines.add('-> p= '+Res);
          Memo1.Lines.add('');
          Edit7.Text:=Res;
        end
      else
        MessageDlg('Impossible de résoudre
l''équation.'+chr(10)+chr(13)+'Veuillez vérifier si toutes les
données nécessaires ont bien été
introduites.',mtWarning,[mbOK],0);
    end
  else if SpeedButton2.Down then
```

```
    begin
      if (Edit2.Text='') and (Edit7.Text<>'') and
(Edit8.Text<>'') then
        begin
          Memo1.Lines.add('p= '+(Edit7.Text));
          Memo1.Lines.add('p''= '+(Edit8.Text));

Res:=floattostrF(1/((1/strtofloat(Edit7.Text))+(1/strtofloat(Ed
it8.Text))),ffGeneral,3,1);
          Memo1.Lines.add('-> f= '+Res);
          Memo1.Lines.add('');
          Edit2.Text:=Res;
        end
      else if (Edit7.Text='') and (Edit2.Text<>'') and
(Edit8.Text<>'') then
        begin
          Memo1.Lines.add('f= '+(Edit2.Text));
          Memo1.Lines.add('p''= '+(Edit8.Text));
          Res:=floattostrF(1/((1/strtofloat(Edit2.Text))-
(1/strtofloat(Edit8.Text))),ffGeneral,3,1);
          Memo1.Lines.add('-> p= '+Res);
          Memo1.Lines.add('');
          Edit7.Text:=Res;
        end
      else if (Edit8.Text='') and (Edit2.Text<>'') and
(Edit7.Text<>'') then
        begin
          Memo1.Lines.add('f= '+(Edit2.Text));
          Memo1.Lines.add('p= '+(Edit7.Text));
          Res:=floattostrF(1/((1/strtofloat(Edit2.Text))-
(1/strtofloat(Edit7.Text))),ffGeneral,3,1);
          Memo1.Lines.add('-> p''= '+Res);
          Memo1.Lines.add('');
          Edit8.Text:=Res;
        end
      else
        MessageDlg('Impossible de résoudre
l''équation.'+chr(10)+chr(13)+'Veuillez vérifier si toutes les
données nécessaires ont bien été
introduites.',mtWarning,[mbOK],0);
    end
  else if SpeedButton3.Down then
    begin
      if (Edit3.Text='') and (Edit2.Text<>'') then
        begin
          Memo1.Lines.add('f= '+(Edit2.Text));

Res:=floattostrF(100/strtofloat(Edit2.Text),ffGeneral,3,1);
          Memo1.Lines.add('-> Pi= '+Res);
          Memo1.Lines.add('');
          Edit3.Text:=Res;
        end
      else if (Edit2.Text='') and (Edit3.Text<>'') then
        begin
          Memo1.Lines.add('Pi= '+(Edit3.Text));

Res:=floattostrF(100/strtofloat(Edit3.Text),ffGeneral,3,1);
          Memo1.Lines.add('-> f= '+Res);
          Memo1.Lines.add('');
          Edit2.Text:=Res;
        end
      else
        MessageDlg('Impossible de résoudre
l''équation.'+chr(10)+chr(13)+'Veuillez vérifier si toutes les
données nécessaires ont bien été
introduites.',mtWarning,[mbOK],0);
    end
  else if SpeedButton4.Down then
    begin
      if (Edit2.Text='') and (Edit4.Text<>'') and
(Edit5.Text<>'') and (Edit6.Text<>'') then
        begin
          Memo1.Lines.add('n= '+(Edit4.Text));
          Memo1.Lines.add('R1= '+(Edit5.Text));
          Memo1.Lines.add('R2= '+(Edit6.Text));
          if strtofloat(Edit5.Text)=0 then Q1:=0 else
Q1:=1/strtofloat(Edit5.Text);
          if strtofloat(Edit6.Text)=0 then Q2:=0 else
Q2:=1/strtofloat(Edit6.Text);
          Res:=floattostrF(1/((strtofloat(Edit4.Text)-1)*(Q1-
Q2)),ffGeneral,3,1);
          Memo1.Lines.add('-> f= '+Res);
          Memo1.Lines.add('');
          Edit2.Text:=Res;
        end
      else if (Edit4.Text='') and (Edit2.Text<>'') and
(Edit5.Text<>'') and (Edit6.Text<>'') then
        begin
          Memo1.Lines.add('f= '+(Edit2.Text));
          Memo1.Lines.add('R1= '+(Edit5.Text));
          Memo1.Lines.add('R2= '+(Edit6.Text));
          if strtofloat(Edit5.Text)=0 then Q1:=0 else
Q1:=1/strtofloat(Edit5.Text);
          if strtofloat(Edit6.Text)=0 then Q2:=0 else
Q2:=1/strtofloat(Edit6.Text);
          Res:=floattostrF(1+(1/(strtofloat(Edit2.Text)*(Q1-
Q2))),ffGeneral,3,1);
          Memo1.Lines.add('-> n= '+Res);
          Memo1.Lines.add('');
          Edit4.Text:=Res;
        end
      else if (Edit5.Text='') and (Edit2.Text<>'') and
(Edit4.Text<>'') and (Edit6.Text<>'') then
        begin
```

```
          Memo1.Lines.add('f= '+(Edit2.Text));
          Memo1.Lines.add('n= '+(Edit4.Text));
          Memo1.Lines.add('R2= '+(Edit6.Text));
          if strtofloat(Edit6.Text)=0 then Q2:=0 else
Q2:=1/strtofloat(Edit6.Text);

Res:=floattostrF(1/((1/(strtofloat(Edit2.Text)*(strtofloat(Edit
4.Text)-1)))+Q2),ffGeneral,3,1);
          Memo1.Lines.add('-> R1= '+Res);
          Memo1.Lines.add('');
          Edit5.Text:=Res;
        end
      else if (Edit6.Text='') and (Edit2.Text<>'') and
(Edit4.Text<>'') and (Edit5.Text<>'') then
        begin
          Memo1.Lines.add('f= '+(Edit2.Text));
          Memo1.Lines.add('n= '+(Edit4.Text));
          Memo1.Lines.add('R1= '+(Edit5.Text));
          if strtofloat(Edit5.Text)=0 then Q1:=0 else
Q1:=1/strtofloat(Edit5.Text);
          Res:=floattostrF(1/((-
1/(strtofloat(Edit2.Text)*(strtofloat(Edit4.Text)-
1)))+Q1),ffGeneral,3,1);
          Memo1.Lines.add('-> R2= '+res);
          Memo1.Lines.add('');
          Edit6.Text:=Res;
        end
      else
        MessageDlg('Impossible de résoudre
l''équation.'+chr(10)+chr(13)+'Veuillez vérifier si toutes les
données nécessaires ont bien été
introduites.',mtWarning,[mbOK],0);
    end
  else
    MessageDlg('Aucune équation n''a été sélectionnée
!',mtWarning,[mbOK],0);
end;

procedure TForm3.SpeedButton6Click(Sender: TObject);
var
  Lens: TEquaLens;
begin
  Lens:=TEquaLens.Create;
  LensNumber:=LensNumber+1;
  RadioGroup1.Items.add('Lentille '+inttostr(LensNumber));
  Lens.LNumber:=LensNumber;
  LensList.add(Lens);
  if RadioGroup1.ItemIndex=-1 then RadioGroup1.ItemIndex:=0;
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
  Top:=337;
  Left:=1;
  LensNumber:=0;
  LensList:=TLENSLIST.Create;
end;

procedure TForm3.SpeedButton7Click(Sender: TObject);
var
  Text:string;
  i,j,k,num: integer;
  Lentille: TLENS;
begin
  if (RadioGroup1.ItemIndex<>-1) then
    begin
      Text:=RadioGroup1.Items[RadioGroup1.ItemIndex];
      RadioGroup1.Items.Delete(RadioGroup1.ItemIndex);
      num:=strtoint(text[10]+text[11]);
      i:=LensList.FindLens(num);
      if TEQUALENS(LensList.Items[i]).Position<>'' then
        begin
          ListeComp.TriComp;

j:=ListeComp.FindComp(strtofloat(TEQUALENS(LensList.Items[i]).P
osition),OLens);
          if j<>-1 then
            begin
              k:=ListeComp.FindTypeComp(OImageOpt);
              if k<>-1 then
                begin
                  Image.Destroy;
                  ListeComp.Delete(k);
                end;
              Lentille:=TLENS(ListeComp.Items[j]);
              Lentille.Destroy;
              Form2.Refresh;
              ListeComp.Delete(j);
            end;
        end;
      TEQUALENS(LensList.Items[i]).Destroy;
      LensList.Delete(i);
      if RadioGroup1.ItemIndex<>-1 then
        begin
          Text:=RadioGroup1.Items[RadioGroup1.ItemIndex];
          num:=strtoint(text[10]+text[11]);
          i:=LensList.FindLens(num);
          Edit1.Text:=TEQUALENS(LensList.Items[i]).Position;
          Edit2.Text:=TEQUALENS(LensList.Items[i]).Focus1;
          Edit3.Text:=TEQUALENS(LensList.Items[i]).Power;
          Edit4.Text:=TEQUALENS(LensList.Items[i]).RefrInd;
          Edit5.Text:=TEQUALENS(LensList.Items[i]).Radius1;
          Edit6.Text:=TEQUALENS(LensList.Items[i]).Radius2;
          Edit7.Text:=TEQUALENS(LensList.Items[i]).PosObj;
```

```
          Edit8.Text:=TEQUALENS(LensList.Items[i]).PosIm;
          Edit9.Text:=TEQUALENS(LensList.Items[i]).Agr;
        end;
    end;
end;

procedure TForm3.RadioGroup1Click(Sender: TObject);
var
  Text: string;
  i,num: integer;
begin
  if RadioGroup1.ItemIndex<>-1 then
    begin
      Text:=RadioGroup1.Items[RadioGroup1.ItemIndex];
      num:=strtoint(text[10]+text[11]);
      i:=LensList.FindLens(num);
      Edit1.Text:=TEQUALENS(LensList.Items[i]).Position;
      Edit2.Text:=TEQUALENS(LensList.Items[i]).Focus1;
      Edit3.Text:=TEQUALENS(LensList.Items[i]).Power;
      Edit4.Text:=TEQUALENS(LensList.Items[i]).RefrInd;
      Edit5.Text:=TEQUALENS(LensList.Items[i]).Radius1;
      Edit6.Text:=TEQUALENS(LensList.Items[i]).Radius2;
      Edit7.Text:=TEQUALENS(LensList.Items[i]).PosObj;
      Edit8.Text:=TEQUALENS(LensList.Items[i]).PosIm;
      Edit9.Text:=TEQUALENS(LensList.Items[i]).Agr;
    end;
end;

procedure TForm3.Edit1Change(Sender: TObject);
var
  Text: string;
  i,num: integer;
begin
  if RadioGroup1.ItemIndex<>-1 then
    begin
      Text:=RadioGroup1.Items[RadioGroup1.ItemIndex];
      num:=strtoint(text[10]+text[11]);
      i:=LensList.FindLens(num);
      if (Sender=Edit1) then
TEQUALENS(LensList.Items[i]).Position:=Edit1.Text;
      if (Sender=Edit2) then
TEQUALENS(LensList.Items[i]).Focus1:=Edit2.Text;
      if (Sender=Edit3) then
TEQUALENS(LensList.Items[i]).Power:=Edit3.Text;
      if (Sender=Edit4) then
TEQUALENS(LensList.Items[i]).RefrInd:=Edit4.Text;
      if (Sender=Edit5) then
TEQUALENS(LensList.Items[i]).Radius1:=Edit5.Text;
      if (Sender=Edit6) then
TEQUALENS(LensList.Items[i]).Radius2:=Edit6.Text;
      if (Sender=Edit7) then
TEQUALENS(LensList.Items[i]).PosObj:=Edit7.Text;
      if (Sender=Edit8) then
TEQUALENS(LensList.Items[i]).PosIm:=Edit8.Text;
      if (Sender=Edit9) then
TEQUALENS(LensList.Items[i]).Agr:=Edit9.Text;
    end;
end;


procedure TForm3.BitCoherClick(Sender: TObject);
begin
  VerifCoher;
end;

function NoBlankValues: boolean;
var
  i: integer;
begin
  Result:=true;
  i:=0;
  with Form3 do
    begin
      if Edit10.Text='' then Result:=false;
      if Edit11.Text='' then Result:=false;
      if Edit12.Text='' then Result:=false;
      if (not RadioButton3.Checked) and (not
RadioButton4.Checked) then Result:=false;
      while (Result=true) and (i<=LensList.count-1) do
        begin
          if TEQUALENS(LensList.Items[i]).Position='' then
Result:=false;
          if (TEQUALENS(LensList.Items[i]).Focus1='')
             and (TEQUALENS(LensList.Items[i]).Power='')
             and ((TEQUALENS(LensList.Items[i]).RefrInd='')
                  or (TEQUALENS(LensList.Items[i]).Radius1='')
                  or
(TEQUALENS(LensList.Items[i]).Radius2=''))
                                  then Result:=false;
          i:=i+1;
        end;
    end;
end;

procedure TForm3.EditTextReset;
begin
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';
  Edit6.Text:='';
  Edit7.Text:='';
```

```
  Edit8.Text:='';
  Edit9.Text:='';
  Edit10.Text:='';
  Edit11.Text:='';
  Edit12.Text:='';
  Memo1.Text:='';
  RadioButton3.Checked:=true;
end;

procedure TForm3.Edit10Click(Sender: TObject);
begin
  TEdit(Sender).SelectAll;
end;

procedure TForm3.BitSimEquaClick(Sender: TObject);
var
  i, LNum: integer;
  Lens: TEquaLens;
begin
  if ListeComp.FindTypeComp(OObjetOpt)=-1 then
    MessageDlg('Vous n''avez pas placé d''objet sur le
banc',mtError,[mbOK],0)
  else
    begin
      if ListeComp.FindTypeComp(OImageOpt)=-1 then
        begin
          Image:=TOBJETOPT.Create(Form2);
          Image.Parent:=Form2;
          Image.TypeComp:=OImageOpt;
          ListeComp.add(Image);
          Form1.SpeedButton1.Down:=true;
          Form1.SpeedButton3.Enabled:=true;
        end;
      CalculIm;
      Edit10.Text:=floattostrf(Objet.Position,ffGeneral,3,1);
      Edit11.Text:=floattostrf(Image.Position,ffGeneral,3,1);
      if Image.Up then
Edit12.Text:=floattostrf(Image.Height/Objet.Height,ffGeneral,2,
1)
         else Edit12.Text:=floattostrf(-
Image.Height/Objet.Height,ffGeneral,2,1);
      if Image.Real then RadioButton3.Checked:=true else
RadioButton4.Checked:=true;
      while RadioGroup1.ItemIndex<>-1 do
RadioGroup1.Items.Delete(0);
      for i:=LensList.count-1 downto 0 do
        begin
          TEQUALENS(LensList.Items[i]).Destroy;
          LensList.Delete(i);
        end;
      LNum:=1;
      for i:=0 to ListeComp.Count-1 do
        begin
          if TCOMP(ListeComp.Items[i]).TypeComp=OLens then
            begin
              Lens:=TEquaLens.Create;
              Lens.LNumber:=LNum;

Lens.Position:=floattostrf(TLENS(ListeComp.Items[i]).Position,f
fGeneral,3,1);

Lens.Radius1:=floattostrf(TLENS(ListeComp.Items[i]).Radius1,ffG
eneral,3,1);

Lens.Radius2:=floattostrf(TLENS(ListeComp.Items[i]).Radius2,ffG
eneral,3,1);

Lens.Focus1:=floattostrf(TLENS(ListeComp.Items[i]).Focus1,ffGen
eral,3,1);

Lens.Power:=floattostrf(1/(TLENS(ListeComp.Items[i]).Focus1/100
),ffGeneral,3,1);

Lens.RefrInd:=floattostrf(TLENS(ListeComp.Items[i]).RefrInd,ffG
eneral,3,1);

Lens.PosObj:=floattostrf(TLENS(ListeComp.Items[i]).Position-
TLENS(ListeComp.Items[i]).PosObj,ffGeneral,3,1);

Lens.PosIm:=floattostrf(TLENS(ListeComp.Items[i]).PosIm-
TLENS(ListeComp.Items[i]).Position,ffGeneral,3,1);

Lens.Agr:=floattostrf(TLENS(ListeComp.Items[i]).Agr,ffGeneral,2
,1);
              LensList.add(Lens);
              RadioGroup1.Items.Add('Lentille
'+inttostr(LNum));
              if RadioGroup1.ItemIndex=-1 then
RadioGroup1.ItemIndex:=0;
              LNum:=LNum+1;
            end;
        end;
    end;
end;

procedure TForm3.BitEquaSimClick(Sender: TObject);
var
  num,i: integer;
  Lentille: TLENS;
  Q1, Q2: Real;
begin
  VerifCoher;
  if Verifcoher then
    begin
```

```
      if ListeComp.FindTypeComp(OObjetOpt)=-1 then
        begin
          Objet:=TObjetOpt.Create(Form2);
          Objet.Parent:=Form2;
          Objet.TypeComp:=OObjetOpt;
          ListeComp.add(Objet);
          Form1.SpeedArrow.Down:=true;
        end;
      Objet.Position:=strtofloat(Edit10.Text);
      Objet.Resize;
      for i:=ListeComp.count-1 downto 0 do
        if TComp(ListeComp.Items[i]).TypeComp=OLens then
          begin
            Lentille:=TLENS(ListeComp.Items[i]);
            Lentille.Destroy;
            ListeComp.Delete(i);
          end;
      for i:=0 to LensList.count-1 do
        begin
              Lentille:=TLens.Create(Form2);
              Lentille.Parent:=Form2;
              ListeComp.add(Lentille);
          if TEQUALENS(LensList.Items[i]).Focus1<>'' then
Lentille.Focus1:=strtofloat(TEQUALENS(LensList.Items[i]).Focus1
)
            else if TEQUALENS(LensList.Items[i]).Power<>'' then
Lentille.Focus1:=100/strtofloat(TEQUALENS(LensList.Items[i]).Po
wer)
          else
              begin
                if
strtofloat(TEQUALENS(LensList.Items[i]).Radius1)=0 then Q1:=0
                  else
Q1:=1/strtofloat(TEQUALENS(LensList.Items[i]).Radius1);
                if
strtofloat(TEQUALENS(LensList.Items[i]).Radius2)=0 then Q2:=0
                  else
Q2:=1/strtofloat(TEQUALENS(LensList.Items[i]).Radius2);

Lentille.Focus1:=1/((strtofloat(TEQUALENS(LensList.Items[i]).Re
frInd)-1)*(Q1-Q2));
              end;
          if (TEQUALENS(LensList.Items[i]).Radius1)='' then
            begin
              Lentille.RefrInd:=1.66;
              Lentille.Radius1:=0;

Lentille.Radius2:=Lentille.Focus1*(Lentille.RefrInd-1);
            end
          else
            begin
Lentille.RefrInd:=strtofloat(TEQUALENS(LensList.Items[i]).RefrI
nd);

Lentille.Radius1:=strtofloat(TEQUALENS(LensList.Items[i]).Radiu
s1);

Lentille.Radius2:=strtofloat(TEQUALENS(LensList.Items[i]).Radiu
s2);
              end;

Lentille.LNumber:=TEQUALENS(LensList.Items[i]).LNumber;

Lentille.Position:=strtofloat(TEQUALENS(LensList.Items[i]).Posi
tion);
            Lentille.Resize;
          end;
      if ListeComp.FindTypeComp(OImageOpt)=-1 then
        begin
          Image:=TObjetOpt.Create(Form2);
          Image.Parent:=Form2;
          Image.TypeComp:=OImageOpt;
          ListeComp.add(Image);
        end;
      CalculIm;
      Form2.Refresh;
      Form1.SpeedButton1.Enabled:=true;
      Form1.SpeedButton1.Down:=true;
    end;
end;

end.
```

unit InfImOpt;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm9 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label4: TLabel;
    Edit2: TEdit;
```

```
    Label5: TLabel;
    Label3: TLabel;
    Edit3: TEdit;
    Label6: TLabel;
    BitBtn2: TBitBtn;
    Label7: TLabel;
    Edit4: TEdit;
    BitBtn3: TBitBtn;
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form9: TForm9;

implementation

uses Simul;

{$R *.DFM}

procedure TForm9.FormShow(Sender: TObject);
begin
  Edit1.Text:=floattostrf(Image.Position,ffGeneral,3,1);
  Edit2.Text:=inttostr(Image.PosY);
  Edit3.Text:=inttostr(Image.Height);
  if Image.Real then Edit4.Text:='réelle' else
Edit4.Text:='virtuelle';
end;

end.
```

unit InfoLens;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm6 = class(TForm)
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    GroupBox1: TGroupBox;
    RadioButton1: TRadioButton;
    Label1: TLabel;
    Edit1: TEdit;
    Label5: TLabel;
    Label6: TLabel;
    Edit2: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    ComboBox1: TComboBox;
    RadioButton2: TRadioButton;
    GroupBox2: TGroupBox;
    Label4: TLabel;
    Label7: TLabel;
    Edit3: TEdit;
    BitBtn3: TBitBtn;
    Label8: TLabel;
    Edit4: TEdit;
    Label9: TLabel;
    BitBtn4: TBitBtn;
    procedure Edit1Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit4Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form6: TForm6;

implementation

{$R *.DFM}

uses Simul, ScrDispl, Banc, Lens, UTypes;

procedure TForm6.Edit1Change(Sender: TObject);
begin
  RadioButton1.Checked:=true;
end;

procedure TForm6.Edit3Change(Sender: TObject);
begin
  RadioButton2.Checked:=true;
end;

procedure TForm6.Edit4Click(Sender: TObject);
begin
  TEDIT(Sender).SelectAll;
end;
```

```
procedure TForm6.BitBtn4Click(Sender: TObject);
var
  Q1,Q2: real;
begin
  if (RadioButton1.Checked) and (Edit1.Text=Edit2.Text)
    then MessageDlg('Si les deux rayons de courbure sont
identiques, la distance focale sera
infinie...',mtError,[mbOK],0)
    else
      begin

TLens(ListeComp.Items[iLensGlob]).Position:=strtofloat(Edit4.Te
xt);
        if RadioButton2.Checked
          then
            begin

TLens(ListeComp.Items[iLensGlob]).Focus1:=strtofloat(Edit3.Text
);
              TLens(ListeComp.Items[iLensGlob]).RefrInd:=1.66;
              TLens(ListeComp.Items[iLensGlob]).Radius1:=0;
              TLens(ListeComp.Items[iLensGlob]).Radius2:=-
TLens(ListeComp.Items[iLensGlob]).Focus1*(TLens(ListeComp.Items
[iLensGlob]).RefrInd-1);
            end
          else
            begin

TLens(ListeComp.Items[iLensGlob]).Radius1:=strtofloat(Edit1.Tex
t);

TLens(ListeComp.Items[iLensGlob]).Radius2:=strtofloat(Edit2.Tex
t);

TLens(ListeComp.Items[iLensGlob]).RefrInd:=strtofloat(ComboBox1
.Text);
              if TLens(ListeComp.Items[iLensGlob]).Radius1=0
                then Q1:=0
                else
Q1:=1/TLens(ListeComp.Items[iLensGlob]).Radius1;
              if TLens(ListeComp.Items[iLensGlob]).Radius2=0
                then Q2:=0
                else
Q2:=1/TLens(ListeComp.Items[iLensGlob]).Radius2;

TLens(ListeComp.Items[iLensGlob]).Focus1:=1/((TLens(ListeComp.I
tems[iLensGlob]).RefrInd-1)*(Q1-Q2));
            end;
        TLens(ListeComp.Items[iLensGlob]).Resize;
        Form2.Refresh;
      end;
end;

end.
```

unit InfoObjOpt;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm7 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    Edit2: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Label4: TLabel;
    Edit3: TEdit;
    Label5: TLabel;
    Label6: TLabel;
    BitBtn4: TBitBtn;
    BitBtn3: TBitBtn;
    procedure FormShow(Sender: TObject);
    procedure Edit1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form7: TForm7;

implementation

uses Banc, ScrDispl, Simul;

{$R *.DFM}

procedure TForm7.FormShow(Sender: TObject);
begin
```

```
  Edit1.Text:=floattostrf(Objet.Position,ffGeneral,3,1);
  Edit2.Text:=inttostr(Objet.PosY);
  Edit3.Text:=inttostr(Objet.Height);
end;

procedure TForm7.Edit1Click(Sender: TObject);
begin
  TEdit(Sender).SelectAll;
end;

procedure TForm7.BitBtn1Click(Sender: TObject);
begin
  Objet.Position:=strtofloat(Edit1.Text);
  Objet.PosY:=round(strtofloat(Edit2.Text));
  Objet.Height:=round(strtofloat(Edit3.Text));
  Objet.Resize;
end;

procedure TForm7.BitBtn4Click(Sender: TObject);
begin
  Objet.Position:=strtofloat(Edit1.Text);
  Objet.PosY:=strtoint(Edit2.Text);
  Objet.Height:=strtoint(Edit3.Text);
  Objet.Resize;
end;

end.
```

unit InfoScreen;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons;

type
  TForm10 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn3: TBitBtn;
    procedure FormShow(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Edit1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form10: TForm10;

implementation

uses Banc, Simul, ScrDispl;

{$R *.DFM}

procedure TForm10.FormShow(Sender: TObject);
begin
  Edit1.Text:=floattostrf(Ecran.Position,ffGeneral,3,1);
end;

procedure TForm10.BitBtn1Click(Sender: TObject);
begin
  Ecran.Position:=strtofloat(Edit1.Text);
  Ecran.Resize;
  if Form1.SpeedButton2.Down then Form5.Paint;
end;

procedure TForm10.Edit1Click(Sender: TObject);
begin
  TEDIT(Sender).SelectAll;
end;

procedure TForm10.BitBtn4Click(Sender: TObject);
begin
  Ecran.Position:=Strtofloat(Edit1.Text);
  Ecran.Resize;
end;

end.
```

unit Lens;

```
interface

uses UTypes, Graphics, Classes, Dialogs, SysUtils;

type
  TLens=class(TCOMP)
```

```
    LNumber: integer;
    Radius1: Real;
    Radius2: Real;
    Focus1: Real;
    RefrInd: Real;
    PosObj: Real;
    PosIm: Real;
    Agr: Real;
    constructor Create(AOwner: TComponent); override;
    procedure Paint; override;
    procedure Resize;
  end;

implementation

uses Banc;

constructor TLens.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Canvas.Pen.Color:=clBlack;
  TypeComp:=OLens;
  Height:=round(Form2.ClientHeight/2);
  Width:=3;
  Height:=1000;
  onMouseMove:=Form2.ObjetMouseMove;
  onMouseDown:=Form2.ObjetMouseDown;
  onMouseUp:=Form2.ObjetMouseUp;
  onDblClick:=Form2.LensDblClick;
end;

procedure TLens.Paint;
var
  Rayon: real;
begin
  inherited Paint;
  Scale:=R.Length/(R.Width-30);
  Canvas.MoveTo(0,0);
  Canvas.LineTo(0,Height);
  Left:=round(Position/scale+10-Form2.HorzScrollBar.Position);
  if (Focus1>=0) then
      begin
        Form2.Canvas.MoveTo(Left,Top);
        Form2.Canvas.LineTo(Left-5,Top+5);
        Form2.Canvas.MoveTo(Left,Top);
        Form2.Canvas.LineTo(Left+5,Top+5);
        Form2.Canvas.MoveTo(Left,Top+Height);
        Form2.Canvas.LineTo(Left-5,Top+Height-5);
        Form2.Canvas.MoveTo(Left,Top+Height);
        Form2.Canvas.LineTo(Left+5,Top+Height-5);
      end
  else if (Focus1<0) then
      begin
        Form2.Canvas.MoveTo(Left,Top);
        Form2.Canvas.LineTo(Left-5,Top-5);
        Form2.Canvas.MoveTo(Left,Top);
        Form2.Canvas.LineTo(Left+5,Top-5);
        Form2.Canvas.MoveTo(Left,Top+Height);
        Form2.Canvas.LineTo(Left-5,Top+Height+5);
        Form2.Canvas.MoveTo(Left,Top+Height);
        Form2.Canvas.LineTo(Left+5,Top+Height+5);
      end;
  Form2.Canvas.Ellipse(round((Position-Focus1)/scale-4-
Form2.HorzScrollBar.Position+10),round(Axe.Top-1)-
4,round((Position-Focus1)/scale+4-
Form2.HorzScrollBar.Position+10),round(Axe.Top-1)+4);
  Form2.Canvas.Ellipse(round((Position+Focus1)/scale-4-
Form2.HorzScrollBar.Position+10),round(Axe.Top-1)-
4,round((Position+Focus1)/scale+4-
Form2.HorzScrollBar.Position+10),round(Axe.Top-1)+4);
end;

procedure TLens.Resize;
begin
  Scale:=R.Length/(R.Width-30);
  Height:=round(Form2.ClientHeight/2);
  Top:=round(Axe.Top-Height/2);
  if Position<=R.Length
    then Left:=round(Position/scale)-
Form2.HorzScrollBar.Position+10
    else Left:=round(R.Length/scale)-
Form2.HorzScrollBar.Position+10;
  if Position<(-10/scale) then Left:=0;

end;


end.
```

unit ListeObjets;

```
interface

uses Classes, Controls,UTypes;

type

    TListeComp=class(TLIST)
      procedure TriComp;
      function FindComp(CompPos: Real; CompType: TTYPEComp):
integer;
```

```
      function FindTypeComp(CompType: TTYPEComp): integer;
      function FindLensNum(number: integer): integer;
      function CountLens: integer;
      function FindNextLens(last: integer): integer;
    end;

    TLENSLIST=class(TLIST)
      function FindLens(Number: integer): integer;
    end;

    function Tri(item1, item2: pointer): integer;


implementation

uses Simul,Banc,EquaLens,Lens;

function Tri(item1, item2: pointer): integer; far;
begin
   if TCOMP(item1).Position<TCOMP(item2).Position then
result:=-1;
   if TCOMP(item1).Position>TCOMP(item2).Position then
result:=1;
   if TCOMP(item1).Position=TCOMP(item2).Position then
result:=0;
end;

procedure TLISTEComp.TriComp;
begin
   sort(Tri);
end;

function TLISTEComp.FindComp(CompPos: Real; CompType:
TTYPEComp): integer;
var
  i: integer;
begin
  Result:=-1;
  i:=0;
  while (Result=-1) and (i<=Count-1) do
    begin
      if (CompPos=TCOMP(Items[i]).Position) and
(CompType=TCOMP(Items[i]).TypeComp) then Result:=i;
      i:=i+1;
    end;
end;

function TLISTEComp.FindLensNum(number: integer): integer;
var
  i: integer;
begin
  Result:=-1;
  i:=0;
  while (Result=-1) and (i<=Count-1) do
    begin
      if (TCOMP(Items[i]).TypeComp=OLens) and
(number=TLENS(Items[i]).LNumber) then Result:=i;
      i:=i+1;
    end;
end;

function TLISTEComp.FindTypeComp(CompType: TTYPEComp): integer;
var
  i: integer;
begin
  Result:=-1;
  i:=0;
  while (Result=-1) and (i<=Count-1) do
    begin
      if TCOMP(Items[i]).TypeComp=CompType then Result:=i;
      i:=i+1;
    end;
end;

function TLISTEComp.CountLens: integer;
var
  i: integer;
  PosEcran,PosObjet: Real;
begin
  Result:=0;
  if Form1.SpeedScreen.Down then PosEcran:=Ecran.Position else
PosEcran:=R.Length;
  if Form1.SpeedArrow.Down then PosObjet:=Objet.Position else
PosObjet:=0;
  for i:=0 to count-1 do
    begin
      if (TCOMP(Items[i]).TypeComp=OLens) and
(TCOMP(Items[i]).Position>=PosObjet) and
(TCOMP(Items[i]).Position<=PosEcran)  then Result:=Result+1;
    end;
end;

function TLISTEComp.FindNextLens(last: integer): integer;
var
  i: integer;
begin
  Result:=-1;
  for i:=last to count-1 do
    begin
      if (TCOMP(Items[i]).TypeComp=OLens) and
(TCOMP(Items[i]).Position>=Objet.Position) and (Result=-1) then
Result:=i;
    end;
end;
```

```
function TLENSLIST.FindLens(Number: integer): integer;
var
  i: integer;
begin
  Result:=-1;
  i:=0;
  while (Result=-1) and (i<=count-1) do
    begin
      if (TEQUALENS(Items[i]).LNumber=Number) then Result:=i
else i:=i+1;
    end;
end;

end.
```

unit ObjetOpt;

```
interface

uses UTypes, Graphics, Classes, Sysutils;

type
  TObjetOpt=class(TCOMP)
    PosY: integer;
    Point: real;
    Pen: TPen;
    Real: boolean;
    Up: boolean;
    constructor Create (AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Paint; override;
    procedure Resize;
    property onDblClick;
  end;

implementation

uses banc,simul,scrdispl,lens;

constructor TObjetOpt.Create(AOWner: TComponent);
begin
  inherited Create(AOwner);
  TypeComp:=OObjetOpt;
  Pen:=TPen.Create;
  Pen.Color:=clBlack;
  Pen.Width:=1;
  Width:=3;
  Height:=30;
  onMouseMove:=Form2.ObjetMouseMove;
  onMouseDown:=Form2.ObjetMouseDown;
  onMouseUp:=Form2.ObjetMouseUp;
  onDblClick:=Form2.ObjOptDblClick;
end;

destructor TObjetOpt.Destroy;
begin
  Pen.Free;
  inherited Destroy;
end;

procedure TObjetOpt.Paint;
begin
  inherited Paint;
  Canvas.Pen:=Pen;
  Canvas.MoveTo(0,0);
  Canvas.LineTo(0,Height);
end;

procedure TObjetOpt.Resize;
begin
  Scale:=R.Length/(R.Width-30);
  Left:=round(Position/scale)-Form2.HorzScrollBar.Position+10;
  Top:=round(Axe.Top-PosY-Height);
end;

end.
```

unit Question;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls, Mask;

type
  TForm8 = class(TForm)
    Label1: TLabel;
    Memo1: TMemo;
    Label2: TLabel;
    Panel1: TPanel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    BitBtn1: TBitBtn;
```

```
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Panel2: TPanel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    MaskEdit1: TMaskEdit;
    MaskEdit2: TMaskEdit;
    MaskEdit3: TMaskEdit;
    MaskEdit4: TMaskEdit;
    MaskEdit5: TMaskEdit;
    MaskEdit6: TMaskEdit;
    procedure FormCreate(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure ResetFields;
    procedure BitBtn1Click(Sender: TObject);
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form8: TForm8;
  MaxQuestNum, QuestNum: integer;

implementation

{$R *.DFM}

uses QuestList;

procedure TForm8.FormCreate(Sender: TObject);
begin
  Left:=620;
  Top:=337;
  WindowState:=wsMinimized;
  QuestNum:=1;
  MaxQuestNum:=3;
  QuestionText(QuestNum);
  BitBtn2.Visible:=false;
  Caption:='Question n°'+inttostr(QuestNum);
end;

procedure TForm8.BitBtn3Click(Sender: TObject);
begin
  QuestNum:=QuestNum+1;
  ResetMaskEdit;
  QuestionText(QuestNum);
  if QuestNum=MaxQuestNum then BitBtn3.Visible:=false;
  BitBtn2.Visible:=true;
  Caption:='Question n°'+inttostr(QuestNum);
end;

procedure TForm8.BitBtn2Click(Sender: TObject);
begin
  QuestNum:=QuestNum-1;
  QuestionText(QuestNum);
  if QuestNum=1 then BitBtn2.Visible:=false;
  BitBtn3.Visible:=true;
  Caption:='Question n°'+inttostr(QuestNum);
end;

procedure TForm8.ResetFields;
begin
  MaskEdit1.Visible:=false;
  MaskEdit2.Visible:=false;
  MaskEdit3.Visible:=false;
  MaskEdit4.Visible:=false;
  MaskEdit5.Visible:=false;
  MaskEdit6.Visible:=false;
  Label3.Visible:=false;
  Label4.Visible:=false;
  Label5.Visible:=false;
  Label6.Visible:=false;
  Label7.Visible:=false;
  Label8.Visible:=false;
  Label9.Visible:=false;
  Label10.Visible:=false;
  Label11.Visible:=false;
  Label12.Visible:=false;
  Label13.Visible:=false;
  Label14.Visible:=false;
end;

procedure TForm8.BitBtn1Click(Sender: TObject);
begin
  QuestionRes(QuestNum);
end;

end.
```

unit QuestList;

```
interface
```

```
uses SysUtils, Dialogs;

procedure QuestionText(i: integer);
procedure QuestionRes(i: integer);
procedure ResetMaskEdit;
function Different(x,y: real): boolean;

var
  QDistFoc, QPosIm: Real;
  QNatIm: string[2];

implementation

uses Question,Banc,Lens,Simul,ObjetOpt, Screen, UTypes,
CalculImage;

procedure ResetMaskEdit;
begin
  with Form8 do
    begin
      MaskEdit1.Text:='';
      MaskEdit2.Text:='';
      MaskEdit3.Text:='';
      MaskEdit4.Text:='';
      MaskEdit5.Text:='';
      MaskEdit6.Text:='';
    end;
end;


procedure QuestionText(i:integer);
begin
  Form8.ResetFields;
  if i=1 then with Form8 do
    begin
      Memo1.Text:='Une lentille sphérique possède deux surfaces
convexes de rayon 0,40 m et 0,60 m. Son indice'+
         ' de réfraction est de 1,50. Trouver sa distance
focale et la position de l''image d''un objet situé à'+
         ' 1,50 m de la lentille.';
      Label3.Visible:=true;
      Label3.Caption:='Dist focale =';
      MaskEdit1.Visible:=true;
      Label4.Visible:=true;
      Label4.Caption:='cm';
      Label5.Visible:=true;
      Label5.Caption:='Pos (%lentille) =';
      MaskEdit2.Visible:=true;
      Label6.Visible:=true;
      Label6.Caption:='cm';
      QDistFoc:=48;
      QPosIm:=70.6;
    end
  else if i=2 then with Form8 do
    begin
      Memo1.Text:='Un objet de 5 cm de diamètre est placé à
33,3 cm d''une lentille divergente de 8 dioptries.'+
         ' Calculer la position de l''image par rapport à la
lentille et sa taille. L''image est-elle réelle ou virtuelle
?';
      Label3.Visible:=true;
      Label3.Caption:='Pos (%lentille) =';
      MaskEdit1.Visible:=true;
      Label4.Visible:=true;
      Label4.Caption:='cm';
      Label5.Visible:=true;
      Label5.Caption:='Image R / V =';
      MaskEdit2.Visible:=true;
      QPosIm:=20;
      QNatIm:='R';
    end
  else if i=3 then with Form8 do
    begin
      Memo1.Text:='Trouver la distance focale d''une lentille
plan-convexe dont le rayon de courbure est de 30 mm'+
         ' et dont l''indice de réfraction vaut 1,50.
Effectuez, à l''aide de cette lentille, un montage formant'+
         ' une image dont l''agrandissement serait de -1,5.';
      Label3.Visible:=true;
      Label3.Caption:='Dist focale =';
      MaskEdit1.Visible:=true;
      Label4.Visible:=true;
      Label4.Caption:='cm';
      QDistFoc:=0.6;
    end;
end;

procedure QuestionRes(i:integer);
var
  Text: string;
  Error, ErrorSim, ErrorEq: boolean;
begin
  Text:='';
  Error:=false;
  if i=1 then with Form8 do

{      Memo1.Text:='Une lentille sphérique possède deux
surfaces convexes de rayon 0,40 m et 0,60 m. Son indice'+
         ' de réfraction est de 1,50. Trouver sa distance
focale et la position de l''image d''un point situé à'+
         ' 1,50 m de la lentille.';}

    begin
      if ListeComp.FindTypeComp(OObjetOpt)=-1 then
```

```
        begin
          Text:=Text+'- Sim: Pas d''objet sur le
banc.'+chr(10)+chr(13)+chr(10)+chr(13);
          Error:=true;
          ErrorSim:=true;
        end;
      if ListeComp.FindTypeComp(OScreen)=-1 then
        begin
          Text:=Text+'- Sim: Pas d''écran sur le
banc.'+chr(10)+chr(13)+chr(10)+chr(13);
          Error:=true;
          ErrorSim:=true;
        end;
      if ListeComp.FindTypeComp(OLens)=-1 then
          begin
            Text:=Text+'- Sim: Pas de lentille sur le
banc.'+chr(10)+chr(13)+chr(10)+chr(13);
            Error:=true;
            ErrorSim:=true;
          end
      else
        if ListeComp.CountLens>1 then
          begin
            Text:=Text+'- Sim: Trop de lentilles sur le
banc.'+chr(10)+chr(13)+chr(10)+chr(13);
            Error:=true;
            ErrorSim:=true;
          end
        else
          begin
            if ListeComp.FindTypeComp(OImageOpt)=-1 then
              begin
                Image:=TOBJETOPT.Create(Form2);
                Image.Parent:=Form2;
                Image.TypeComp:=OImageOPt;
              end;
            CalculIm;
            if
Different(TLENS(ListeComp.Items[ListeComp.FindTypeComp(OLens)])
.Focus1,QDistFoc) then
              begin
                Text:=Text+'- Sim: La lentile n''a pas la bonne
distance focale.'+chr(10)+chr(13)+chr(10)+chr(13);
                Error:=true;
                ErrorSim:=true;
              end;
            if
Different(TLENS(ListeComp.Items[ListeComp.FindTypeComp(OLens)])
.Position-Objet.Position,150) then
              begin
                Text:=Text+'- Sim: La position de l''objet
n''est pas correcte.'+chr(10)+chr(13)+chr(10)+chr(13);
                Error:=true;
                ErrorSim:=true;
              end;
            if ListeComp.FindTypeComp(OScreen)<>-1 then
            if
Different(TSCREEN(ListeComp.Items[ListeComp.FindTypeComp(OScree
n)]).Position-
TLENS(ListeComp.Items[ListeComp.FindTypeComp(OLens)]).Position,
QPosIm) then
              begin
                Text:=Text+'- Sim: La position de l''écran
n''est pas correcte.'+chr(10)+chr(13)+chr(10)+chr(13);
                Error:=true;
                ErrorSim:=true;
              end;
          end;
      if Different(strtofloat(MaskEdit1.Text),QDistFoc) then
        begin
          Text:=Text+'- Val Num: La distance focale n''est pas
correcte.'+chr(10)+chr(13)+chr(10)+chr(13);
          Error:=true;
          ErrorEq:=true;
        end;
      if Different(strtofloat(MaskEdit2.Text),QPosIm) then
        begin
          Text:=Text+'- Val Num: La position de l''image n''est
pas correcte.'+chr(10)+chr(13);
          Error:=true;
          ErrorEq:=true;
        end;
      if Error then
        begin
         if MessageDlg('Votre réponse n''est pas
correcte'+chr(10)+chr(13)+'Voulez-vous plus de
détails?',mtWarning,[mbYes,mbNo],0)=mrYes then
            MessageDlg(Text,mtWarning,[mbOK],0)
        end
      else MessageDlg('Votre réponse est correcte
!'+chr(10)+chr(13)+'Vous pouvez passer à un autre
exercice.',mtInformation,[mbOK],0);
    end
  else if i=2 then with Form8 do
    begin
      if Different(strtofloat(MaskEdit1.Text),QPosIm) then
        begin
          Text:=Text+'- La position de l''image n''est pas
correcte.'+chr(10)+chr(13)+chr(10)+chr(13);
          Error:=true;
        end;
      if (MaskEdit2.Text<>QNatIm) then
        begin
```

```
          Text:=Text+'- La nature de l''image n''est pas
correcte.'+chr(10)+chr(13)+chr(10)+chr(13);
          Error:=true;
        end;
      if Error then MessageDlg(Text,mtWarning,[mbOK],0)
        else MessageDlg('Votre réponse est correcte
!'+chr(10)+chr(13)+'Vous pouvez passer à un autre
exercice.',mtInformation,[mbOK],0);
    end
  else if i=3 then with Form8 do
    begin

    end;
end;

function Different(x,y: Real): boolean;
begin
  Result:=false;
  if (x<y-abs(0.5*y)/100) or (x>y+abs(0.5*y)/100) then
Result:=true;
end;

end.
```

unit Ruler;

```
interface

uses Controls, Graphics, Classes, Dialogs, SysUtils;

type
  TRuler=class(TGraphicControl)
    Length: Real;
    Pen: TPen;
    Brush: TBrush;
    Font: TFont;
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Paint; override;
    procedure Resize;
  end;

implementation

uses Banc;

constructor TRuler.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Pen:=TPen.Create;
  Pen.Color:=clWhite;
  Brush:=TBrush.Create;
  Brush.Color:=clGray;
  Font:=TFont.Create;
  Font.Color:=clWhite;
  Height:=30;
end;

destructor TRuler.Destroy;
begin
  Pen.Free;
  Brush.Free;
  Font.Free;
  inherited Destroy;
end;

procedure TRuler.Paint;
var
  pos,pas, Interval: Integer;
  scale: Real;
begin
  inherited Paint;
  Canvas.Pen:=Pen;
  Canvas.Brush:=Brush;
  Canvas.Font:=Font;
  Canvas.Rectangle(0,0,Width,Height);
  Interval:=max-min;
  if Interval<=60 then pas:=5
    else if Interval<=200 then pas:=10
      else if Interval<=400 then pas:=20
        else pas:=50;
  scale:=(Width-30)/Length;
  pos:=min;
  while pos<=Length do
    begin
      Canvas.MoveTo(10+round(pos*scale),0);
      Canvas.LineTo(10+round(pos*scale),10);
      Canvas.TextOut(7+round(pos*scale),15,IntToStr(pos));
      Canvas.MoveTo(10+round(pos*scale+scale*pas/2),0);
      Canvas.LineTo(10+round(pos*scale+scale*pas/2),5);
      pos:=pos+pas;
    end;
end;

procedure TRuler.Resize;
begin
  Width:=Form2.ClientWidth*round(length/(max+1));
  Top:=Form2.ClientHeight-Height;
end;
```

```
end.
```

unit ScrDispl;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, Arrow, Axis;

type
  TForm5 = class(TForm)
    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure Paint;
  end;

var
  Form5: TForm5;
  Fleche: TArrow;
  AxeDispl: TAxis;

implementation

uses Banc, Simul, ObjetOpt, UTypes, CalculImage;

{$R *.DFM}

procedure TForm5.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  if ListeComp.FindTypeComp(OObjetOpt)<>-1 then Fleche.Destroy;
  Form1.SpeedButton2.Down:=false;
  Form1.SpeedButton2.Hint:='Activer la visualisation de
l''écran';
end;

procedure TForm5.Paint;
begin
  Height:=Ecran.Height;
  AxeDispl.Pen.Color:=clAqua;
  AxeDispl.Pen.Mode:=pmNotXor;
  AxeDispl.Top:=round(ClientHeight/2)-4;
  AxeDispl.Width:=ClientWidth;
  if ListeComp.FindTypeComp(OImageOpt)=-1 then
    begin
      Image:=TObjetOpt.Create(Form2);
      Image.Parent:=Form2;
      Image.Width:=0;
      Image.TypeComp:=OImageOpt;
      ListeComp.add(image);
      ListeComp.TriComp;
    end;
  CalculIm;
  if ListeComp.FindTypeComp(OObjetOpt)<>-1 then
    begin
      if Image.Up then
        begin
          Fleche.Direction:=adUp;
          Fleche.Height:=round(Image.Height);
          Fleche.Top:=AxeDispl.Top-Image.PosY-Image.Height;
        end
      else
        begin
          Fleche.Direction:=adDown;
          Fleche.Height:=round(Image.Height);
          Fleche.Top:=AxeDispl.Top-Image.PosY-Image.Height;
        end;
      if (abs(round(Ecran.Position-Image.Position))<=20) and
(Image.Real=true) then
        begin
          Fleche.Pen.Width:=abs(round(Ecran.Position-
Image.Position))+1;
          Fleche.Pen.Color:=$00FFFFFF-
($00111111*Fleche.Pen.width);
        end
      else
        begin
          Fleche.Pen.Width:=0;
          Fleche.Pen.Color:=clBlack;
        end;
      Fleche.Paint;
    end;
end;

procedure TForm5.FormCreate(Sender: TObject);
begin
  AxeDispl:=TAxis.Create(Form5);
  AxeDispl.Parent:=Form5;
  AxeDispl.Top:=round(ClientHeight/2);
  AxeDispl.Width:=ClientWidth;
  AxeDispl.Paint;
end;

end.
```

unit Screen;

```
interface

uses Controls, Classes, Graphics, SysUtils, Dialogs, UTypes;

type
  TScreen=class(TCOMP)
    Pen: TPen;
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure Paint; override;
    procedure Resize;
    property onMouseMove;
    property onMouseDown;
    property onMouseUp;
    property onDblClick;
  end;

implementation

uses Banc;

constructor TScreen.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  TypeComp:=OScreen;
  Pen:=TPen.Create;
  Pen.Color:=clPurple;
  Pen.Width:=6;
  Width:=6;
  Height:=round(Form2.ClientHeight/1.5);
  Top:=round((Form2.ClientHeight/2)-(Height/2));
  onMouseMove:=Form2.ObjetMouseMove;
  onMouseDown:=Form2.ObjetMouseDown;
  onMouseUp:=Form2.ObjetMouseUp;
  onDblClick:=Form2.ScreenDblClick;
end;

destructor TScreen.Destroy;
begin
  Pen.Free;
  inherited Destroy;
end;

procedure TScreen.Paint;
begin
  inherited Paint;
  Canvas.Pen:=Pen;
  Canvas.MoveTo(0,0);
  Canvas.LineTo(0,Height);
end;

procedure TScreen.Resize;
begin
  Scale:=R.Length/(R.Width-30);
  if Height>=round(Form2.ClientWidth/3) then
Height:=round(Form2.ClientWidth/3);
  Top:=round((Form2.ClientHeight/2)-(Height/2));
  Left:=round(Position/scale)-Form2.HorzScrollBar.Position+10;
end;

end.
```

unit Simul;

```
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  Buttons, ExtCtrls, Menus, ComCtrls, UTypes, StdCtrls,
ObjetOpt, EquaLens,
  ListeObjets;

type

  TForm1 = class(TForm)
    SpeedPanel: TPanel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Exit1: TMenuItem;
    Options1: TMenuItem;
    Zoomin1: TMenuItem;
    ZoomOut1: TMenuItem;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    SpeedScreen: TSpeedButton;
    SpeedArrow: TSpeedButton;
    SpeedLens: TSpeedButton;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    Mode1: TMenuItem;
    Problme1: TMenuItem;
    EquationsSimulateur1: TMenuItem;
    SimulateurEquations1: TMenuItem;
    Libre1: TMenuItem;
    SpeedProb: TSpeedButton;
    SpeedLibre: TSpeedButton;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
```

```
    SpeedButton3: TSpeedButton;
    UpDown1: TUpDown;
    Edit1: TEdit;
    SpeedZoomIn: TSpeedButton;
    SpeedZoomOut: TSpeedButton;
    SpeedMeasure: TSpeedButton;
    Aide1: TMenuItem;
    procedure SpeedZoomInClick(Sender: TObject);
    procedure SpeedZoomOutClick(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Zoomin1Click(Sender: TObject);
    procedure ZoomOut1Click(Sender: TObject);
    procedure SpeedMeasureClick(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure SpeedScreenClick(Sender: TObject);
    procedure SpeedArrowClick(Sender: TObject);
    procedure SpeedLensClick(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedProbClick(Sender: TObject);
    procedure SpeedLibreClick(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure UpDown1Click(Sender: TObject; Button:
TUDBtnType);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  CompPut: TTypeComp;
  Image: TObjetOpt;
  ListeComp: TLISTEComp;

implementation

{$R *.DFM}

uses Banc, Screen, Arrow, DLens, Lens, ScrDispl, Question,
Equation;

procedure TForm1.SpeedZoomInClick(Sender: TObject);
begin
  Zoomin1Click(Sender);
end;

procedure TForm1.SpeedZoomOutClick(Sender: TObject);
begin
  ZoomOut1Click(Sender);
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
  ListeComp.Destroy;
  Halt;
end;

procedure TForm1.Zoomin1Click(Sender: TObject);
var
  i: integer;
begin
  max:=round(max/1.55);
  R.Resize;
  Axe.Resize;
  Form2.Refresh;
  for i:=0 to ListeComp.count-1 do
    begin
      case TCOMP(ListeComp.items[i]).TypeComp of
        OScreen: TSCREEN(ListeComp.items[i]).Resize;
        OObjetOpt:  TOBJETOPT(ListeComp.items[i]).Resize;
        OLens: TLENS(ListeComp.items[i]).Resize;
        OImageOpt: TOBJETOPT(ListeComp.Items[i]).Resize;
      end;
    end;
  SpeedZoomIn.Enabled:=(max>=20);
  ZoomIn1.Enabled:=(max>=20);
  SpeedZoomOut.Enabled:=(max<=150);
  ZoomOut1.Enabled:=(max<=150);
  Form2.Refresh;
end;

procedure TForm1.ZoomOut1Click(Sender: TObject);
var
  i: integer;
begin
  max:=round(max*1.55);
  R.Resize;
  Axe.Resize;
  Form2.Refresh;
  for i:=0 to ListeComp.count-1 do
    begin
      case TCOMP(ListeComp.items[i]).TypeComp of
        OScreen: TSCREEN(ListeComp.items[i]).Resize;
        OObjetOpt:  TOBJETOPT(ListeComp.items[i]).Resize;
        OLens: TLENS(ListeComp.items[i]).Resize;
        OImageOpt: TOBJETOPT(ListeComp.Items[i]).Resize;
      end;
    end;
  SpeedZoomIn.Enabled:=(max>=20);
  ZoomIn1.Enabled:=(max>=20);
  SpeedZoomOut.Enabled:=(max<=150);
```

```
  ZoomOut1.Enabled:=(max<=150);
  Form2.Refresh;
end;

procedure TForm1.SpeedMeasureClick(Sender: TObject);
var
  Button: TMouseButton;
  Shift: TShiftState;
  X, Y: Integer;
begin
  SpeedMeasure.Enabled:=false;
  Form2.Cursor:=crCross;
  Form2.Canvas.Pen.Style:=psDot;
end;

procedure TForm1.FormResize(Sender: TObject);
begin
  StatusBar1.Panels[0].Width:=ClientWidth-220;
end;

procedure TForm1.SpeedScreenClick(Sender: TObject);
var
  i: integer;
begin
  if Form2.Cursor<>crDefault then
SpeedScreen.Down:=not(SpeedScreen.Down) else
     begin
       if SpeedScreen.Down then
         begin
           Form2.Cursor:=crUpArrow;
           CompPut:=OScreen;
           SpeedScreen.Hint:='Enlever l''écran du banc';
         end
       else
         begin
           SpeedScreen.Hint:='Positionner un écran sur le
banc';
           Ecran.Destroy;
           Form2.Refresh;

i:=ListeComp.FindComp(Ecran.Position,Ecran.TypeComp);
           ListeComp.Delete(i);
           Form5.Visible:=false;
           SpeedButton2.Down:=false;
           SpeedButton2.Enabled:=false;
         end;
     end;
end;

procedure TForm1.SpeedArrowClick(Sender: TObject);
var
  i: integer;
begin
  if (Form2.Cursor<>crDefault) then
SpeedArrow.down:=not(SpeedArrow.Down) else
     begin
       if SpeedArrow.Down then
         begin
           Form2.Cursor:=crUpArrow;
           CompPut:=OObjetOpt;
           SpeedArrow.Hint:='Enlever l''objet (flèche) du
banc';
         end
       else
         begin
           SpeedArrow.Hint:='Positionner un objet (flèche) sur
le banc';
           Objet.Destroy;
           i:=ListeComp.FindTypeComp(OObjetOpt);
           ListeComp.Delete(i);
           if SpeedButton1.Down then
             begin
               SpeedButton1.Down:=false;
               SpeedButton1.Enabled:=false;
               SpeedButton3.Down:=false;
               SpeedButton3.Enabled:=false;
               Image.Destroy;
               i:=ListeComp.FindTypeComp(OImageOpt);
               ListeComp.Delete(i);
             end;
           Form2.Refresh;
         end;
     end;
end;

procedure TForm1.SpeedLensClick(Sender: TObject);
begin
  if Form2.Cursor=crDefault then
    begin
      Form4.Showmodal;
      if Form4.ModalResult=mrCancel then
Form2.Cursor:=crDefault;
    end;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  Flech: TArrow;
  PosObjOpt, i: integer;
  CompFound: boolean;
begin
  SpeedButton3.Enabled:=not(SpeedButton3.Enabled);
  if (ListeComp.Count<>0) and (SpeedButton1.Down) then
    begin
```

```
      SpeedButton3.Down:=true;
      SpeedButton1.Hint:='Enlever le dessin de l''image';
      ListeComp.TriComp;
      if TCOMP(ListeComp.Items[0]).TypeComp<>OObjetOpt then
        begin
          if MessageDlg('L''objet ne se trouve pas à l''extrème
gauche du banc.'+chr(10)+chr(13)+'Voulez-vous tout de même
continuer?',

mtInformation,[mbYes,mbNo],0)=mrYes then
            begin
              if ListeComp.FindTypeComp(OImageOpt)=-1 then
                begin
                  Image:=TObjetOpt.Create(Form2);
                  Image.Parent:=Form2;
                  Image.Width:=3;
                  Image.TypeComp:=OImageOpt;
                  Image.onDblClick:=Form2.ImOptDblClick;
                  ListeComp.add(image);
                  ListeComp.TriComp;
                  Form2.Refresh;
                end
              else
                Image.Width:=3;
            end
          else
              SpeedButton1.Down:=false;
              SpeedButton1.Hint:='Dessiner l''image';
        end
      else
        begin
                SpeedButton3.Down:=false;
          if ListeComp.FindTypeComp(OImageOpt)=-1 then
            begin
              Image:=TObjetOpt.Create(Form2);
              Image.Parent:=Form2;
              Image.Width:=3;
              Image.TypeComp:=OImageOpt;
              Image.onDblClick:=Form2.ImOptDblClick;
              ListeComp.add(image);
              Form2.Refresh;
            end
          else
            Image.Width:=3;
        end;
      if ListeComp.FindTypeComp(OLens)=-1 then Image.Width:=0;
    end
  else if (ListeComp.Count<>0) and (not SpeedButton1.Down) then
    begin
      SpeedButton1.Hint:='Dessiner l''image';
```

```
TObjetOpt(ListeComp.Items[ListeComp.FindTypeComp(OImageOpt)]).D
estroy;
      ListeComp.Delete(ListeComp.FindTypeComp(OImageOpt));
      Form2.Refresh;
    end;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  if SpeedButton2.Down then
    begin
      ListeComp.TriComp;
      SpeedButton2.Hint:='Désactiver la visualisation de
l''écran';
      if ListeComp.FindTypeComp(OObjetOpt)<>-1 then
      begin
        Fleche:=TArrow.Create(Form5);
        Fleche.Parent:=Form5;
      end;
      Form5.Visible:=true;
      Form5.Paint;
    end
  else
    begin
      SpeedButton2.Hint:='Activer la visualisation de
l''écran';
      if ListeComp.FindTypeComp(OObjetOpt)<>-1 then
Fleche.Destroy;
      Form5.Visible:=false;
    end;
end;

procedure TForm1.SpeedProbClick(Sender: TObject);
begin
  Form8.WindowState:=wsNormal;
  Form8.BitBtn1.Enabled:=true;
  Form8.BitBtn2.Enabled:=true;
  Form8.BitBtn3.Enabled:=true;
  SpeedButton1.Visible:=false;
  SpeedButton2.Visible:=true;
  SpeedButton3.Visible:=false;
  UpDown1.Visible:=false;
  Edit1.Visible:=false;
  SpeedLens.Visible:=true;
  SpeedScreen.Visible:=true;
  SpeedArrow.Visible:=true;
  Form3.BitEquaSim.Enabled:=false;
  Form3.BitSimEqua.Enabled:=false;
  BitBtn2Click(Sender);
end;
```

```
procedure TForm1.SpeedLibreClick(Sender: TObject);
begin
  Form8.WindowState:=wsMinimized;
  Form8.BitBtn1.Enabled:=false;
  Form8.BitBtn2.Enabled:=false;
  Form8.BitBtn3.Enabled:=false;
  SpeedButton1.Visible:=true;
  SpeedButton1.Enabled:=false;
  SpeedButton2.Visible:=true;
  SpeedButton2.Enabled:=false;
  SpeedButton3.Visible:=true;
  SpeedButton3.Enabled:=false;
  UpDown1.Visible:=true;
  Edit1.Visible:=true;
  Form3.BitEquaSim.Enabled:=true;
  Form3.BitSimEqua.Enabled:=true;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
var
  i: integer;
  Lentille: TLENS;
begin
  if SpeedButton2.Down then
    begin
      Form5.Close;
    end;
  while Form3.RadioGroup1.ItemIndex<>-1 do
    begin
      Form3.RadioGroup1.Items.Delete(0);
    end;
  for i:=LensList.count-1 downto 0 do
    begin
      TEQUALENS(LensList.Items[i]).Destroy;
      LensList.Delete(i);
    end;
  i:=ListeComp.FindTypeComp(OObjetOpt);
  if i<>-1 then
    begin
      Form1.SpeedArrow.Down:=false;
      Objet.Destroy;
      ListeComp.Delete(i);
    end;
  i:=ListeComp.FindTypeComp(OImageOpt);
  if i<>-1 then
    begin
      Image.Destroy;
      ListeComp.Delete(i);
    end;
  i:=ListeComp.FindTypeComp(OScreen);
  if i<>-1 then
    begin
      Form1.SpeedScreen.Down:=false;
      Ecran.Destroy;
      ListeComp.Delete(i);
    end;
  for i:=ListeComp.Count-1 downto 0 do
    begin
      Lentille:=TLENS(ListeComp.Items[i]);
      Lentille.Destroy;
      ListeComp.Delete(i);
    end;
  Form2.Refresh;
  if SpeedLibre.Down then
    begin
      SpeedButton1.Visible:=true;
      SpeedButton1.Down:=false;
      SpeedButton2.Visible:=true;
      SpeedButton2.Down:=false;
    end;
  Form3.EditTextReset;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  if SpeedButton3.Down then
   begin Form2.Refresh; end
   else Form2.Refresh; begin end;
end;

procedure TForm1.UpDown1Click(Sender: TObject; Button:
TUDBtnType);
begin
  Form2.Refresh;
end;

end.
```

unit UTypes;

```
interface

uses Classes, Controls;

type

  TTypeComp=(OScreen, OObjetOpt, OImageOpt, OLens);

  TCOMP=class(TGRAPHICCONTROL)
    TypeComp: TTYPEComp;
    Position: real;
```

```
  end;
implementation

end.
```