INSA ROUEN
À taille humaine,
à l'échelle du monde

Projet de Physique P6-3
STPI/P6-3/2010 – 048

VISUALISATION DE MOUVEMENTS NEWTONIENS

Etudiants :
Robin THOMAS
Jean CREUSEFOND
Étienne CARLES

Enseignant-responsable du projet :
Yves MONTIER

Date de remise du rapport : ***18/06/2010***

Référence du projet : ***STPI/P6-3/2010 – 048***

Intitulé du projet : ***Mouvements Newtoniens***

Type de projet : ***Simulation***

Objectifs du projet :

Simuler de façon informatique les mouvements newtoniens. Notre projet s'oriente plus particulièrement sur les mouvements d'un satellite autour d'une planète mais aussi ceux des systèmes à deux étoiles. On illustre ainsi la deuxième loi de Kepler (loi des aires) et le fait que considérer un référentiel basé sur une planète ne soit pas forcément barycentrique.

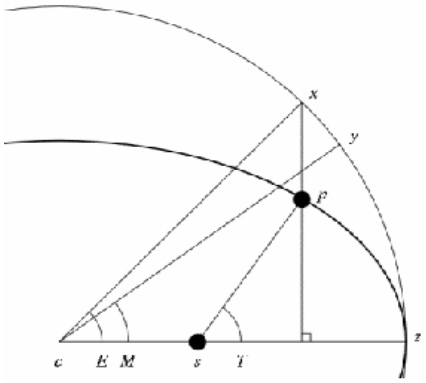
Notons que notre programme devra être conçu de manière à être visuellement clair, il sera en effet utilisé par Mr Montier pour son cours de P2 (mécanique du point).

INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE ROUEN
Département Sciences et Techniques Pour l'Ingénieur
BP 8 – place Emile Blondel - 76131 Mont-Saint-Aignan - tél : 33 2 35 52 83 00 - fax : 33 2 35 52 83 69

TABLE DES MATIÈRES

NOTATIONS, ACRONYMES

Anomalie excentrique : Dans la description de l'orbite képlerienne d'un objet céleste, l'anomalie excentrique , en général notée E, est l'angle entre la direction du périapse et la position courante d'un objet sur son orbite, projetée sur le cercle exinscrit perpendiculairement au grand axe de l'ellipse, mesuré au centre de celle-ci)



Source : projet 35, année 2008/09

Variable : En programmation, une variable est une zone mémoire de taille fixe de l'ordinateur, servant à stocker une valeur, un caractère, un objet... En java, la création d'une variable se fait en annonçant son type (double = réel, int = entier, …) puis son nom. Par exemple, l'instruction double x = 2 crée la variable x, qui est un réel, et lui donne la valeur 2.

Panel : Sur une fenêtre, il s'agit d'une couche graphique sur laquelle on peut « dessiner » (changer la couleur des pixels).

1. INTRODUCTION

Les lois du mouvement de Newton sont à la base de sa grande théorie concernant le mouvement des corps. Ces lois, fondées sur le principe de relativité des mouvements ainsi que sur la loi de la gravitation universelle, permettent d'interpréter aussi bien la chute des objets que le mouvement de la Lune autour de la Terre.

Dans ce projet nous avions pour but de modéliser deux mouvements Newtoniens : le mouvement d'un satellite autour d'une planète et celui d'un système à étoile double. Les étoiles doubles sont un ensemble de deux étoiles suffisamment rapprochées pour que l'on puisse observer des effets gravitationnels entre elles. Ce phénomène est très intéressant a étudier, car il permet de visualiser un référentiel non galiléen représenté par le centre de gravité des deux étoiles.

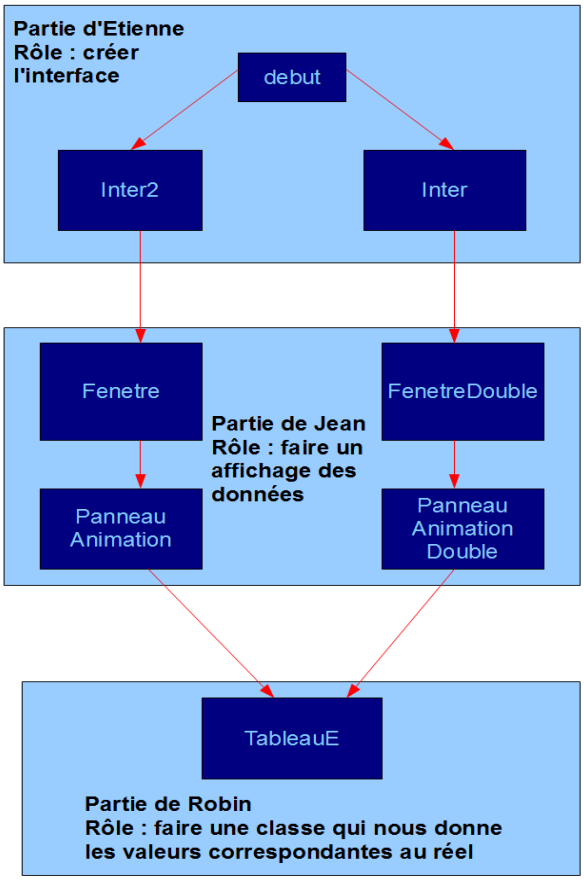
Notons que la première animation n'est qu'un cas particulier de la seconde. Une planète est tout simplement bien plus lourde que l'autre, ce qui implique qu'elle est très proche du centre de gravité des deux centres pondérés par leur masse. L'intérêt de séparer les animations est de pouvoir illustrer la deuxième loi de Kepler.

Notre projet ne consiste pas toutefois à établir les lois qui permettent de décrire ces différents mouvements, notre démarche consistant en effet à travailler sur les résultats démontrés par nos prédécesseurs dans ce projet. Nous devrons donc nous les approprier pour développer la meilleure solution de modélisation.

2. MÉTHODOLOGIE / ORGANISATION DU TRAVAIL

Le diagramme suivant résume comment le travail a été réparti.

Chaque case bleue foncée correspond à un classe de notre programme.
Chaque flèche rouge correspond à une communication/un appel entre classes.



3. TRAVAIL RÉALISÉ ET RÉSULTATS

3.1. Détermination et exploitation des données théoriques

3.1.1. Partie théorique

La théorie physique en tant que telle n'était pas le but premier de ce projet car cela avait été déjà réalisé par nos prédécesseurs (cf annexe) . Toutefois pour créer les différentes animations, il nous fallait parfaitement maitriser les résultats théoriques afin de ne pas commettre d'erreurs sur la simulation.

Notre programme a pour fonction de simuler les déplacements de deux points matériels M_1 et M_2 de masse m_1 et m_2 .Nous avons convenu de coder l'intégralité du programme en java pour lui permettre de gérer en même temps le calcul et la partie graphique. En revanche, cela posait un problème théorique, celui de devoir calculer les coordonnées en cartésien.

Trajectoire elliptique

Nous avons alors défini certaines données : r_0 la distance initiale, v_0 la vitesse initiale, $m1$ et $m2$. Ainsi nous avons pu calculer l'excentricité de notre ellipse (e) ainsi que ses caractéristiques a et b en fonction des conditions initiales.

$$e=1-\sqrt[3]{\frac{4\pi r_0^3}{GT^2(m_1+m_2)}} \quad a=\frac{r_0}{1-e} \quad b=a\sqrt{1-e^2}$$

De plus, la troisième lois de Kepler nous donne la période de rotation en fonction des conditions initiales :

$$T=\sqrt{\frac{4\pi r_0^3}{G(m_1+m_2)(1-e)^3}}$$

Ainsi nous avons pu calculer toutes les coordonnées des points placés sur l'ellipse, sans que le facteur temps n'intervienne. Pour l'introduire nous avons utilisé l'équation de Kepler qui permet de calculer l'anomalie excentrique

$$\frac{2\pi}{T}(t-t_p)=E-esinE$$

Ce qui nous permet ensuite de déterminer les positions en x et en y du satellite sur l'ellipse :

$$x=a(cosE-e) \quad y=a\sqrt{1-e^2}sinE$$

Système à deux corps

Pour le système à deux corps, le calcul de e et E reste le même, les coordonnées sont données par :

$$x_1=\frac{-m_2 r_0 a}{m_1+m_2}(cosE-e) \quad y_1=\frac{-m_2 r_0 a}{m_1+m_2}\sqrt{1-e^3}sinE$$

$$x_2=\frac{m_1 r_0 a}{m_1+m_2}(cosE-e) \quad y_2=\frac{m_1 r_0 a}{m_1+m_2}\sqrt{1-e^3}sinE$$

3.1.2. Exploitation des données : partie mathématique

Dans cette partie la principale difficulté a été de résoudre l'équation de Kepler car elle ne peut pas l'être algébriquement. Cette équation est donnée en fonction de e , T ,et de t .

Résolution de l'équation de Kepler

$$\frac{2\pi}{T}(t-t_p)=E-esinE$$

Nous avons alors, choisi d'utiliser la méthode par dichotomie qui consiste à trouver la valeur approchée du zéro d'une fonction sur un intervalle $[a;b]$ donné, à condition que:

- $f(a)<0$

- $f(b)>0$

-f continue sur $[a;b]$

Pour cela dans un premier temps on teste le signe de $f\left(\frac{a+b}{2}\right)$:

-si $f\left(\frac{a+b}{2}\right)$ <0, cela implique que le zéro de la fonction se situe sur $\left[\frac{a+b}{2};b\right]$ nous pouvons ainsi définir cet intervalle comme intervalle de recherche.

-sinon cela implique que le zéro de la fonction se trouve sur $\left[a;\frac{a+b}{2}\right]$ qui peut alors se servir de cet intervalle de recherche.

Cet algorithme a l'avantage de pouvoir ajuster la précision des valeurs en mettant une condition sur la largeur de l'intervalle de recherche.

De plus il est totalement adaptable à notre problème en posant

$$F(E,t,T)=E-esinE-2\pi\frac{t}{T}=0$$

$$\text{On a t}\in]0;T[\ etT>0$$

$$F(0,t,T)=0-2\pi\frac{t}{T}<0\ car\ \frac{t}{T}>0$$

$$F(2\pi,t,T)=2\pi-2\pi\frac{t}{T}>0\ car\ \frac{t}{T}<1$$

$$\text{De plus}$$

$$\dot{F}(E,t,T)=1-e\,\text{cosE est defini pourE}\in[0;2\pi]$$

On peut donc approximer E pour tout t sur la période.

Ainsi nous avons pu recommencer cette opération un certain nombre de fois sur l'intervalle, suivant la précision voulue. Une fois E calculé, nous avons aisément déterminé les cordonnées.

3.1.3. Partie programmation

Le java permet de faire les calculs grâce à des fonctions comme la suivante :

```
public double calcul_e(double m1,double m2,double v0,double r0)
{
        double G = 6.673E-11;
```

```
        double e=Math.abs(1-(r0*v0*v0/(G*(m1+m2))));
        return e;
}
```

Cette fonction a pour argument m_1 , m_2 , v_0 et r_0 , il permet de calculer e puis de le retourner.

Ainsi nous pouvons calculer e , T , a et b . Ainsi a et b sont envoyés dans le programme graphique pour dessiner l'ellipse, puis e et T sont utilisés pour résoudre l'équation de Kepler. Tout d'abord nous avons créé la fonction $F(x,t,T)$:

```
 public double F_de_E(double x,double t,double T)
{
        double F =x-e*Math.sin(x)-(2*Math.PI*t/T);
        return F;
}
```

Ainsi nous pouvons résoudre l'équation pour un instant t donné.

Cf annexe : tableau_E ligne 53 à 79

Cet opération est renouvelée sur 500 valeurs de la période. Nous avons pu créer deux tableaux associant les valeurs de T et de E . Enfin il nous a fallut créer un moyen de récupérer une valeur de E quelque soit la valeur de t .

```
public double getE(double t)
{
       for(int i = 0;i<499;i++)
       {
             if(t>=valeurs_de_t[i]&&t<valeurs_de_t[i+1])
                   return valeurs_de_E[i];
        }
       return 0;
```

Cette méthode permet de récupérer la valeur E la plus proche de t .

Ainsi nous pouvons calculer les positions des différents astres en fonction de nos conditions initiales. Elles représentent bien les mouvements des corps célestes selon les lois de la mécanique. Après une adaptation pour les rendre utilisables (modification des période, et des distances) elle seront affichées dans l'animation.

3.2. Partie animation

L'animation est le fait de prendre les valeurs renvoyées par la classe contenant les formules mathématiques pour les afficher sur un Panel. La base pour simuler un mouvement est connue et facile à implémenter. Par contre, d'autres points furent plus ardus : l'affichage et le calcul des aires ainsi que la conversion des données réelles en données affichables.

3.2.1. L'Animation, l'Univers et le Reste

Comment réaliser une animation informatique? Il faut savoir que des langages tels que java sont très éloignés de la machine(1). Néanmoins, java reste un langage de programmation générale : il ne permet pas de créer une animation simplement. Il faut pour cela utiliser un concept vieux comme le monde du cinéma : au delà d'un flux d'images de 24/ s, l'œil du spectateur ne vois plus une succession d'images, mais un mouvement.

C'est pourquoi notre classe fenêtre appelle la méthode incrementer() du panneau toutes les 10 ms (via un Timer(2)). Dans cette méthode, le Panel ira changer légèrement ses variables correspondant aux coordonnées des points à afficher. Ce changement se fera bien sûr suivant le modèle théorique. Ensuite, encore dans la fenêtre, la méthode repaint() du panneau sera appelée, qui en fait exécutera paintComponant de la classe du panneau. Dans cette méthode, le panneau se voit être recouvert d'un carré blanc afin que la couche précédente ne soit plus visible. Ensuite, les points sont dessinés aux coordonnées correspondantes, et enfin le dessin des autres choses à afficher se réalise : ellipse et aires pour l'animation de Kepler, les vecteurs et le centre de gravité pour l'autre animation.

Quelles parties du code java est concernée? Comme expliqué précédemment, nous devons regarder à deux endroits différents : la classe gérant la fenêtre (soit Fenetre et FenetreDouble) ainsi que la classe gérant le Panel (PanneauAnimation et PanneauAnimationDouble). Les deux animations fonctionnent selon le même concept, le code est donc le même à ce niveau, à la différence près que les éléments à afficher sont différents.

Dans la classe gérant la fenêtre, on trouvera donc un Timer, gérant la boucle infinie. Il sera initialisé lors de l'appel de `t = new Timer(10, new TimerListener());`. Le contenu de la boucle infinie sera quant à lui défini dans la `class TimerListener`.

Cf annexe : PanneauAnimation et PanneauAnimationDouble

Dans les panneaux, nous rencontrerons la méthode suivante :

```
protected void paintComponent(Graphics g)
```

Rappelons que cette méthode sera appelée à chaque fois qu'un repaint() sera invoqué. Cette méthode servira à « peindre » sur le panneau des formes simples grâce à l'objet Graphics.

Un point sur cet objet : il s'agit de l'élément qui gère l'affichage de l'élément dans lequel nous sommes, ici le panneau. Par exemple, le dessin du fond se fera grâce à la ligne suivante :

```
g.fillRect(0, 0, width, height);
```

L'objet g dessinera donc un rectangle rempli dont la coordonnée de départ sera le point en haut à gauche du panneau, et qui sera des mêmes dimensions que celui-ci (même hauteur et même largeur).

Nous avons donc vu comment une animation se créait. Nous arrivons maintenant sur la plus grande difficulté rencontrée : le dessin de l'aire parcourue.

---

(1) Un langage éloigné de la machine implique qu'il dispose d'instructions complexes, très éloignées du matériel. Par exemple : en java, créer un fenêtre se fait grâce à l'instruction new JFrame(), alors qu'elle requière énormément d'instructions au niveau matériel

(2) Timer = objet qui réalise les mêmes instructions selon une fréquence donnée (boucle infinie)

3.2.2. Le dessin et calcul de l'aire : un problème pas si simple

1 Les tableaux, une étape indispensable

Résumons les données disponibles : grâce à la partie mathématique de Robin, nous avons une ellipse ainsi qu'un mouvement. Il est indispensable de définir arbitrairement deux intervalles temporels égaux durant lesquels l'aire s'affiche. Comment transformer tout ceci en données exploitables?

Comme nous voulons une aire qui s'affiche au fur et à mesure que le satellite est dans l'intervalle temporel, il est clair qu'il nous faut trouver les coordonnées des points que parcourra celui-ci lors de cet intervalle. De plus, comme nous ne désirons pas un programme trop lourd, la recherche de ces points se fera lors du lancement du programme, et non au fur et à mesure.

Voici donc comment nous avons procédé : nous divisons l'intervalle temporel en 500 points (correspondant ici à la variable *precision*) et sur chaque sous-intervalle nous recherchons les coordonnées associées au mouvement du satellite. Les valeurs trouvées remplissent quatre tableaux : il nous faut les coordonnées en X et en Y des points pour chaque aire.

Si vous voulez avoir plus d'informations, vous pouvez vous rendre au début de la méthode initTableaux() de la classe PanneauAnimation qui implémente la logique ci-dessus.

Maintenant que nous avons ces tableaux, il nous faut deux choses : calculer les valeurs numériques de l'aire quand le satellite passera sur chaque point, et dessiner la fameuse aire.

2 Le calcul de l'aire : un peu de mathématiques

Maintenant que nous avons une découpe de l'intervalle temporel de dessin de l'aire, il nous faut qu'à chaque coordonnée soit attribuée une aire qui corresponde à l'aire déjà dessinée lorsque le satellite atteint le point. L'objectif de ceci est qu'au fur et à mesure que l'aire se dessine, la valeur de l'aire correspondante s'affiche.

La formule principale pour faire ceci est la suivante (n étant l'indice du point étudié) :

$$A_n = \int_{\theta=\theta_0}^{\theta_n} \frac{\rho^2 \, d\theta}{2}$$

θ_n ici correspond à l'angle formé entre la base (G, Ux, Uy) et le point d'indice n

θ_0 est l'angle du premier point de l'aire

Il nous faut donc trouver une fonction nous renvoyant ce fameux angle. Après quelques recherches et tests, la fonction suivante fonctionne (accessible dans PanneauAnimation):

```
public double getAngleEllipse(int posX,int posY)
```

Pour son contenu, veuillez vous reporter à la ligne 228 de PanneauAnimation.

Il nous faut aussi une fonction calculant rhô :

```
private double distanceEllipse(int posX, int posY)
{
    return Math.sqrt((posX-xCentre)*(posX-xCentre)+(posY-yCentre)*(posY-yCentre));
}
```

Une fois ces deux outils crées, nous approximons l'intégrale en une somme des points déjà trouvés :

$$A_n = \sum_{i=1}^{n} \frac{\rho^2 \left(\theta_i - \theta_{i-1}\right)}{2}$$

L'implémentation informatique d'une telle somme serait lourde, mais on peut remarquer la chose suivante :

$$A_n = A_{n-1} + \frac{\rho^2 \left(\theta_n - \theta_{n-1}\right)}{2}$$

Ce qui est bien plus simple, car il suffit de parcourir une seule fois le tableau des coordonnées. L'implémentation se fait sur une ligne, mais quelle ligne ! Il suffit, pour chaque case de notre nouveau tableau, récupérer la case précédente et d'utiliser les méthodes vues ci-dessus. Pour l'implémentation en Java, veuillez vous référer à la ligne 204 de la classe PanneauAnimation (toujours dans la méthode initTableaux()).

Notons toutefois que les aires relevées sont en pixels carrés. Nous utiliserons donc la résolution écran de l'utilisateur pour connaître son nombre de dpi (points par pouce) afin de convertir le tout en cm^2.

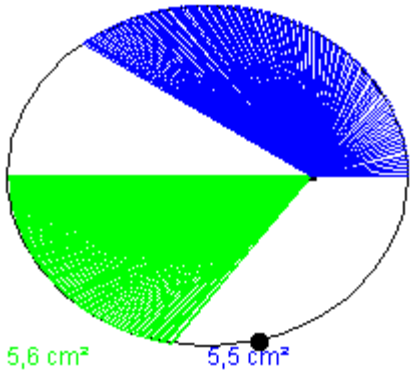
Notre objectif est atteint : pour chaque point que nous dessinerons pendant l'intervalle temporel, nous aurons une aire associée que nous pourrons afficher. Un point reste obscur : comment afficher cette aire?

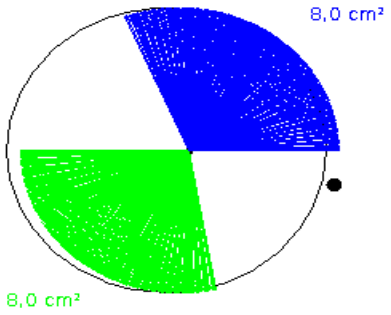
3 L'affichage de l'aire

Java nous permet donne énormément d'outils graphiques afin de mener à bien cette opération. Notre premier réflexe a d'ailleurs été de chercher une méthode nous simplifiant la tâche. Malheureusement, aucune ne permet de réaliser ce que nous cherchons à faire directement. Nous avons donc dû réfléchir au problème suivant : comment, grâce à des formes géométriques simples, dessiner l'aire d'une portion d'ellipse?

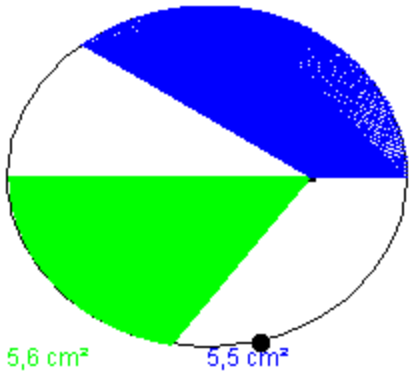
Nous avons des tableaux de points correspondant aux positions successives du satellite. En dessinant un trait entre chaque position et le centre, le problème semble résolu. Nous pouvons même l'améliorer en reliant à chaque point de l'arc son point opposé (le point n sera relié à 1, le point n-1 sera relié à 2 etc.). Mais en réalité, le résultat est imparfait. Voici

une image du résultat en utilisant d'abord la première méthode, puis les deux simultanément :





Notez l'apparition de pixels blancs dans les aires. Ceux-ci sont dus au fait que nous faisons des lignes à partir de pixels, qui sont par définition carrés ou rectangulaires. Comme ces lignes ne sont ni horizontales, ni verticales, des pixels restent non dessinés. Une méthode pour éliminer une grande partie de ces points est le dessin d'un quadrilatère qui a pour points : le centre, le premier point de l'aire, le dernier, et celui du milieu. Nous obtenons le résultat suivant :



Un meilleur résultat serait difficile à obtenir. Nous en resterons donc là

Pour ce qui est de l'implémentation, vous trouverez tout le code à la méthode dessinDeLaire de PanneauAnimation.

3.2.3. Le lien entre le modèle réel et l'informatique : les coefficients

Sous ce titre énigmatique se cache une réalité bien simple : si les valeurs du modèle mathématique ne sont pas converties, nous aurons des périodes de l'ordre d'une dizaine d'années et nous aurions besoin d'écrans de quelques milliers de kilomètres de côté pour pouvoir avoir un aperçu de ce qui se passe. Il est donc évident qu'il faut appliquer des coefficients à ces valeurs. La question est : lesquels?

Un coefficient dépend des paramètres d'affichage : nous l'avons déjà évoqué, il proviens du fait que les aires s'expriment en pixels carrés, et que nous voulons des

centimètres. Nous avons à notre disposition la résolution de l'écran de l'utilisateur ainsi que le taux de conversion cm en pouce. C'est tout ce que nous avons besoin de savoir.

Il y a *résolution* pixels par pouce. En partant du principe que les pixels sont carrés, il y a donc $résolution^2$ pixels par $pouce^2$. Un pouce fait 2.54 cm, donc un $pouce^2$ fait 2.54^2 , ce qui fait que la formule précédente donne :

$$1\ pixel = \frac{2.54^2}{resolution^2}\ cm^2$$

Notre coefficient est maintenant déterminé.

Passons maintenant au coefficient spatial : pour ce faire, nous fixons un paramètre de l'ellipse comme de taille constante informatiquement. Elle devra toujours être égale à 100 pixels. Fort de ce paramètre, tout le reste en découle : à chaque longueur que nous recevrons du modèle, nous la multiplierons par 100/a (ici, il s'agit évidemment du a réel). Cette constante s'appelle dans le programme *rapportDistance*.

Nous devons enfin trouver un dernier coefficient, cette fois temporel. En partant sur une période informatique de 5s, et d'un taux de rafraichissement de 10ms, il suffit de la période réelle pour trouver ce coefficient. En effet, si on veut que l'animation fasse une boucle de 5s, cela veut dire que l'ensemble de la période doit être parcourue en 5s, donc en 500 pulsations du programme. L'incrémentation du temps réel (correspondant dans le programme à la variable t) toutes les 10ms doit donc d'être de période/500(variable *vitesse*). Ceci est notre dernier coefficient.

3.3. Création de l'interface interactivité avec l'utilisateur

En informatique, une interface est un dispositif qui permet des échanges et interactions entre différents acteurs. L'interface qui est présentée à l'utilisateur est nommée interface utilisateur. Elle donne accès aux fonctions du programme par le biais d'un clavier ou d'une souris (périphérique d'entrée) tout en les représentant d'une manière graphique sur un écran (périphérique de sortie). On peut aussi appeler cela le couplage entre l'homme et la machine. L'un des buts de l'interface est ainsi de donner des outils permettant ainsi à l'utilisateur d'interagir plus agréablement ou plus efficacement avec la machine.

3.3.1. Quels outils à mettre à disposition de l'utilisateur ?

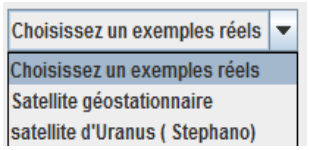
Pour nos deux animations, nous avons réussi à décrire les mouvements newtoniens grâce à seulement quatre paramètres, à savoir, la masse du premier astre, la masse du second, la distance les séparant ainsi que la vitesse initiale au niveau de la périhélie. Afin d'exploiter pleinement notre programme informatique, l'utilisateur doit être dans la possibilité de faire varier ces paramètres. En effet, d'une part, cela lui permettra de pouvoir étudier des différents cas de figure possibles qui puissent exister en fonction de l'excentricité. Rappelons que cette dernière dépend seulement des quatre paramètres précédents.

Nous avons donc décidé de mettre au service de l'utilisateur plusieurs choix: des curseurs, des exemples réels prédéfinis ou laisser l'usager libre de choisir ses propres valeurs. Un bouton « OK » a aussi été inséré pour lancer l'application. Comme nous avions

deux applications : loi des aires et système à étoile double, nous avons du créer deux classes (respectivement « Inter2 » et « Inter ») qui donnent un interface utilisateur avec les choix précédents. Rappelons que l'excentricité doit être comprise entre 0 et 1 afin d'avoir une forme elliptique.

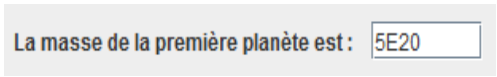
Les curseurs : On fixe la masse du premier astre à 7E24 et on détermine les valeurs minimums et maximums des autres paramètres dans le but d'avoir une excentricité correcte. Donc en faisant varier ces derniers, dans la totalité des plages de valeurs proposées, à l'aide des curseurs, l'excentricité sera obligatoirement comprise entre 0 et 1. Lorsque l'on choisi une valeur avec un curseur, cette dernière est directement envoyée dans une zone de texte (« JTextField » en JAVA) représentant la valeur actuelle. Cette dernière affiche les valeurs prises par l'emplacement du curseur dès que celui-ci bouge. Quand l'utilisateur a décidé d'une valeur précise, il appuie sur le bouton « OK ». A ce moment l'information se trouvant dans « valeurActuelle » est envoyé en paramètre des classes fenetre ou fenetreDouble, correspondant respectivement au lancement de l'application de la loi des aires ou de celle du système à double étoile. Le fonctionnement des autres curseurs est exactement le même.

Les exemples réels prédéfinis : On a semblé intéressant pour l'utilisateur de pouvoir visionner des cas de figures réels comme un satellite géostationnaire autour de la terre ou la terre autour du soleil par exemple. Pour cela on initialise un « JComboBox », ce qui est tout simplement une liste déroulante comme on a l'habitude d'utiliser. Grâce à la réactivité des



boutons, lorsque l'usager choisi parmi les exemples possibles, les « JTextField » (ou zone de texte représentant les valeurs actuelles) sont remplie adéquatement en fonction du choix. En effet, si l'on choisi « Satellite géostationnaire », le champ de la valeur actuelle représentant la masse du premier astre sera égale à 6E24 (masse de la Terre) , le deuxième champ(masse du satellite) : 282, le troisième (distance entre le satellite et la Terre) : 42E6 et le quatrième champ (vitesse initiale à laquelle le satellite est lancé à la périhélie) : 3E3. Comme pour les curseur, lors de l'appui sur « OK » les valeurs se trouvant dans les « JtextField » sont envoyés dans les paramètre de fenetre ou fenetreDouble. Il est évident que pour ces exemple prédéfinis l'excentricité est toujours comprise entre 0 et 1.

Le libre choix des valeurs : Avec les curseurs et les exemples prédéfinis l'utilisateur à un large éventail de valeurs, mais il existe encore une infinité de combinaisons entre les paramètres pour lesquels 0 < e < 1. C'est pour cela que l'usager peut rentrer les valeurs qu'il désirent dans les « JtextField » (zone de texte). Après avoir remplis toutes les champs de



texte, l'utilisateur appuie sur « OK » pour lancer l'application de la même manière qu'avec les outils précédents.

L'année dernière, l'animation ne pouvait marcher que pour certaines valeurs bien définies. Pour une excentricité non comprise entre 0 et 1, leur logiciel ne fonctionnait pas.

Pour notre programme, lorsque l'on clique sur le bouton « OK », le programme fais un test pour calculer l'excentricité avant de lancer la modélisation. Si celle-ci est supérieure à 1, alors un message d'information est affiché signalant que l'excentricité était trop élevée, et l'animation ne se lance pas. Dans ce cas, l'utilisateur peut entrer de nouveau des valeurs et regarder si ce coup-ci l'animation peut se lancer.

3.3.2. Les grilles, un gestionnaire de placement efficace mais pas si simple.

Un gestionnaire de placement est un algorithme qui, appliqué à un container (notre panneau/fenêtre) va permettre de positionner les différents composants qu'on y ajoute. Dans cette grille virtuelle, toutes les cellules d'une même ligne auront toujours la même hauteur. De même, chacune des cellules d'une même colonne auront toute la même largeur.

Les différentes possibilités de placement des grilles :

a - Les propriétés gridx et gridy : Ces propriétés permettent de positionner un composant dans la grille. La numérotation de la grille commence en gridx et gridy égales à zéro dans le coin supérieur gauche. Ces deux propriétés ne peuvent contenir que des valeurs entières supérieures ou égales à zéro. En se déplaçant de gauche à droite, gridx augmente de 1 par cellule. En se déplaçant vers le bas, gridy augmente de 1 par cellule.

b - Les propriétés gridwidth et gridheight : Ces propriétés permettent de préciser le nombre de cases qu'occupera un composant horizontalement (gridwidth) et verticalement (gridheight). Ces propriétés doivent être une valeur entière supérieure ou égale à 1.

c - La propriété anchor : Nous l'avons vu précédemment, un composant est positionné sur une cellule de la grille grâce à sa position x et y et qu'il peut s'étendre sur plusieurs cellules aussi bien horizontalement que verticalement. La propriété anchor permet de spécifier un point d'ancrage à un composant à l'intérieur de sa (ou ses) cellule(s). Par défaut, le composant sera centré horizontalement et verticalement dans les cellules qui lui sont allouées. En effet, si la taille de la case de la grille est supérieure aux dimensions du composant, nous pourrons choisir où ancrer ce composant au sein de cet espace. Il est possible de spécifier des constantes comme NORTH, EAST, SOUTHEAST.

f - Les propriétés weightx et weighty :

Ces deux propriétés permettent de définir comment l'espace supplémentaire sera distribué parmi les composants horizontalement (weightx) et verticalement (weighty). Par défaut, aucun poids n'est défini. Ces deux propriétés sont très subtiles et, mal utilisées, peuvent provoquer des désastres au niveau de l'interface que nous développons. Les valeurs les plus communes sont les valeurs 0 et 1. Le problème avec ces deux propriétés est qu'il peuvent provoquer des désastres au niveau de l'interface, ce qui rend le placement des composants plutôt difficile. En effet les composants restent centrés dans la fenêtre.

4. CONCLUSIONS ET PERSPECTIVES

4.1. Conclusions personnelles

4.1.1. Robin

J'ai beaucoup apprécié ce projet puisque j'ai eu l'impression d'avoir une réelle démarche de recherche et de développement. De plus, le fait que l'on aie eu des objectifs clairement définis dès le début du projet et le fait de reprendre un travail de recherche déjà effectuer m'ont beaucoup stimulé. J'ai été particulièrement motivé par le fait que ce projet ait un réel but puisque les programmes devraient être montrés aux élèves de première année dans le but d'illustré le cours de P2.

Tout cela m'a permis de m'impliquer dans le projet mais aussi de garder ma motivation tout au long de sa mise en place.

Durant ce projet mon rôle été d'utiliser les résultats du projet de l'année précédente, puis de programmer une méthode pour les exploiter. Je me plaçais donc à l'amont du travail des autres membres. Ce fut une position difficile car sans moi certaines parties du projet ne pouvaient pas avancer.

Toutefois dans l'ensemble ce projet est une réussite car nous avons selon moi atteint les objectifs sans le stress habituel des fins de projet mais aussi car il a permis d'enrichir mes compétences en informatique et en gestion de projet.

4.1.2. Jean

Ce projet a été constructif et intéressant, et ce pour 3 raisons :

D'abord, il nous a permis d'approfondir nos connaissances théoriques sur le sujet, en nous donnant un bagage de culture générale.

Ensuite, nous avons étés obligés de travailler en équipe. Cela signifie énormément de choses : répartition des rôles, obligation de délais en cas d'interdépendances... Cet aspect sera probablement une grande part de notre vie d'ingénieurs.

Enfin, un projet informatique est toujours un certain challenge. La recherche de solutions aux nombreux problèmes qui se sont posés a été très enrichissante.

4.1.3. Étienne

Ce projet m'a permis d'explorer plus en détails les mouvements newtoniens à travers deux modélisations : l'illustration de la loi des aires et les mouvements d'un système à étoile double. L'étude théorique étant déjà effectuée l'an dernier, je me suis enrichi au niveau des connaissances informatiques dans le langage Java.

J'avais un rôle qui était très indépendant des autres puisque je m'occupais de l'interface de notre programme. Avant de relier les parties, je pouvais avancer sans le soutien des travaux effectués par le reste du groupe. J'ai trouvé ce rôle très intéressant car cela m'a permis une assez grande liberté dans mon travail tout en conservant un lien étroit avec mes équipiers pour l'avancement cohérent du projet.

Selon moi, ce projet est une satisfaction puisque les objectifs décidés au début ont été atteints. En effet, grâce à un bon équilibre planning – travail, nous avons su tirer le meilleur du travail de groupe afin de finir dans les temps.
En conclusion, je dirai que tout le monde a dû et su apporter du sien pendant ce semestre. Cela nous a permis de surmonter les moments difficiles quand ça ne marchait pas comme on le voulait, et de connaître des moments de réussite quand on arrivait à faire exactement ce que l'on voulait avec notre programme.

En conclusion, je dirai que tout le monde a dû apporter du sien pendant ce semestre et qu'il y a eu des moments difficiles où ça ne marchait pas comme on le voulait, comme des moments de réussite où l'on réussissait à faire exactement ce que l'on voulait avec notre programme.

4.2. Perspectives pour la poursuite de ce projet

Il s'est avéré après un test de dernière minute que les paramètres correspondants à Pluton faisaient inexplicablement bugger l'animation. Il serait sans doute intéressant de se pencher sur ce problème : d'où viens-t-il? Problème d'animation, de modèle mathématique, de variable informatique?

Autre bug persistant et inexpliqué : parfois, le satellite ne se déplace pas tout à fait sur l'ellipse (le cas « satellite géostationnaire » parle de lui-même). Ce genre de soucis viens probablement d'une série d'approximations ou d'un manque de précision du modèle mathématique. Ces deux problèmes méritent que l'on se penche dessus. Malheureusement, par manque de temps et d'effectifs, nous ne pouvons cette année tenter de les résoudre.

Ensuite, un autre point pourrait s'avérer complexe à mettre en œuvre : la possibilité d'afficher des trajectoires hyperboliques, telles que celle des comètes. Montrer par l'animation que la deuxième loi de Kepler fonctionne toujours dans ces cas rendrait sans doute les choses plus claires pour les élèves.

5. BIBLIOGRAPHIE

Tutoriel de Java : http://www.siteduzero.com/tutoriel-3-10601-programmation-en-java.html
Valide à la date du 18/06/10

Wikipedia

Etoiles doubles : http://fr.wikipedia.org/wiki/Étoile_binaire

Templates : http://fr.wikipedia.org/wiki/(134340)_Pluton (par exemple)

La méthode dichotomique : http://fr.wikipedia.org/wiki/Dichotomie
L'équation de Kepler : http://fr.wikipedia.org/wiki/R%C3%A9solution_de_l%27%C3%A9quation_de_Kepler

Valide à la date du 18/06/10

6. ANNEXES (NON OBLIGATOIRE)

6.1. Listings des programmes réalisé

Voici une liste exhaustive des classes réalisées dans notre programme :

Main, debut, inter, inter2, Fenetre, FenetreDouble, PanneauAnimation, PanneauAnimationDouble et TableauE.

Le contenu de ces classes étant trop lourd pour ce projet, nous l'avons temporairement stocké en ligne à l'adresse suivante :

http://www.megaupload.com/?d=NOJEP855

Il se peut qu'il y ait eu quelques changements mineurs entre la version archivée et la version finale.

Pour lire ces fichiers, nous conseillons d'utiliser un logiciel adapté tel que NotePad, gedit ou même eclipse si vous désirez compiler. En effet, ces logiciels permettent (parfois en cherchant dans les options) de faire ressortir certaines parties du code.

6.2. Projet de l'année précédente

Le rapport du projet de l'année 08/09 est disponible à l'adresse suivante :

https://moodle.insa-rouen.fr/file.php/165/Archives_annees_precedentes/2008/Rapport_P6-3_2009_35.pdf