Introduction

Ce travail pratique a été donné par le cours de Simulation Discrète et le but de lui est : de réaliser un modèle de simulation en C ou C++ permettant de reproduire le fonctionnement de système d'un atelier de traitement de surfaces ; de proposer un moyen simple pour déterminer les dates d'entrées des pièces permettant d'obtenir une solution sans dépassement de temps de séjour ; de réaliser une animation graphique simple.

La simulation est une technique de modélisation informatique du monde réel, ça est: développer un programme informatique qui représente et qui remplace le système réel, ainsi on peut le comprendre, l’analyser, l’expérimenter et l’améliorer.

Pour présenter le travail on a organisé le rapport de la façon suivante : D'abord on va faire une description du système, après on va présenter les structures et les fonctions qui on a créé pour rouler le programme, ensuite on va afficher les résultats qui on a obtenu et pour fin faire une brève conclusion.

Description du Système :

En générale on considère que le système de production est composée de 12 cuves, auquel la cuve 1 est la cuve d'entrée et la cuve 12 est la cuve de sortie. Chaque pièce a une gamme de production et elle doit respecter les cuves de sa gamme. Dans cette système il y a un robot qui fait les transports des les pièces. Ce robot a des temps de déplacement d'une cuve à l'autre qui peuvent êtres estimées par la formule suivante : ***tij = (i + j)/ 4***où l i et j sont le numéro des cuves i et j. On estime à une unité de temps la durée de montée et la durée de descente.

Autre caractéristique du système est que le numéro de pièces dans le système est limité par une constate Nmax. Dans notre simulation la pièce n'entre que dans la cuve 1 quand le numéro de pièces est inférieur que le Nmax. Et la cuve 1 a priorité de traitement sur les autres cuves. Pour les autres cuves (a exception de la cuve 12), le robot associe une priorité de traitement donnée par la formule suivante :

***Priorite = t – tmax***

***tmax – tmin***

Auquel le *t* est comme le temps courant du robot, *tmax* est le temps maximum de sortie de la pièce de la cuve et *tmin* est le temps minimum de sortie de la pièce.

Le programme que on a développé permit que le utilisateur choisit trois types de systèmes:(1) Système aléatoire;(2) système auquel le utilisateur va mettre la ordre des pièces ; e pour fin (3) le Système qui trouve une bonne ordre de pièces.

Pour faire le système 3 on a proposer une couplage entre le algorithme de descende qui a été décrit par Jourdan (2003) et le modèle précédent. Si l'utilisateur choisir le système (2) il pourra mettre l'ordre d'entrée de 6 pièces, mais avec les systèmes 2 et 3 le utilisateur pourra choisir la quantité de pièces jusqu'à 50.

Après quelques testes on a vu qu'il y a la possibilité de blocage du système, ça s'arrive quand il y a une pièce dans la Cuve i qui doit aller à Cuve j, mais dans la Cuve j il y a une pièce que doit aller pour Cuve i, ainsi le robot ne peut pas traiter ces pièces et le programmes va arrêter. Alors on a donner au utilisateur la possibilité de interdire la blocage.

Une autre chose que peut avoir lieu est une pièce rester dans la cuve i après la date maximale de la cuve. Alors cette pièce va avoir une retard. Dans notre programme le utilisateur a la opportunité de interdire les retard aussi.

Structures

Pour faire le programme on génère différents structures. Ces structures sont :

« **T\_Operation** » qui contient le numéro de cuve et le temps minimum et maximum de permanence dans cette cuve. Cette structure va être utile pour remplir le vecteur de la structure T\_gamme.

« **T\_Robot**» qui stocke le temps nécessaire pour le robot monter et descendre dans chaque cuve, une matrice avec la distance entre chaque cuve, la date du prochain événement et la position actuelle du robot. Le Robot est la machine principale du programme, puis est lui le responsable pour tous les mouvements pièces et pour le changement du temps courant ;

«  **t\_entree** » qui contient le numéro de la gamme de la pièce que va entrer dans le système;

**« T\_Ordre** » contenant la quantité de pièces du système et un vecteur qui stocke l'ordre d'entrée des pièces;

**«T\_Gamme** »  qui contient le nombre d'opérations d'une gamme et un vecteur contenant l'ordre de ses opérations. Cette structure va stocker les informations du fichier de lecture ;

**«T\_Piece** » Est la représentations de la pièce et qui contient la gamme de cette pièce, le numéro de la gamme, un identificateur, son retard dans le processus et son date d’entrée et de sortie du système;

**«T\_Cuve** » Est la représentation des cuves et pour ça elle va stocker l'état de la cuve (0 vide, 1 pleine), la date d'entrée dans la cuve, la date minimale et maximale pour sortir et la pièce qui est dans la cuve;

**«T\_File** » qui stock le nombre de pièces dans la file et un vecteur des pièces. Ces structure va être le file de sortie de notre programme;

**«T\_Solution** » Est la structures qui nous donne le résultat finale du programme. Elle a une file de sortie, un vecteur d’ordre, la durée totale du système et le retard total.

Fonctions

Pour faire ce travail on a eu besoin de construire des fonctions pour changer quelque paramètre, imprimer des résultats, initialiser les variables, et faire le système rouler. Dans notre travail il y a 32 types de fonctions par conséquence on ne va que parler des fonctions principales. Alors on parlera des fonctions : Lire\_Fichier(); simuler (), Rouler\_Systeme().

## Lire\_Fichier()

Cette fonction faire la lectures des instances. Pour rouler le système il faut avoir les gammes de chaque pièces. Dans la feuille qui a les orientations du travail il y a la description de 6 types de gammes. Dans chaque gamme on a une la ordre de opérations et chaque opération a le numéro de la cuves et la duré minimale et maximale que la pièce peut rester dans la cuve. Pour lire les instances on a copié tous les informations de ce document et les met dans un fichier créé pour nous que s'appelle Teste.txt.

Au-dessous il y a une photo du fichier :

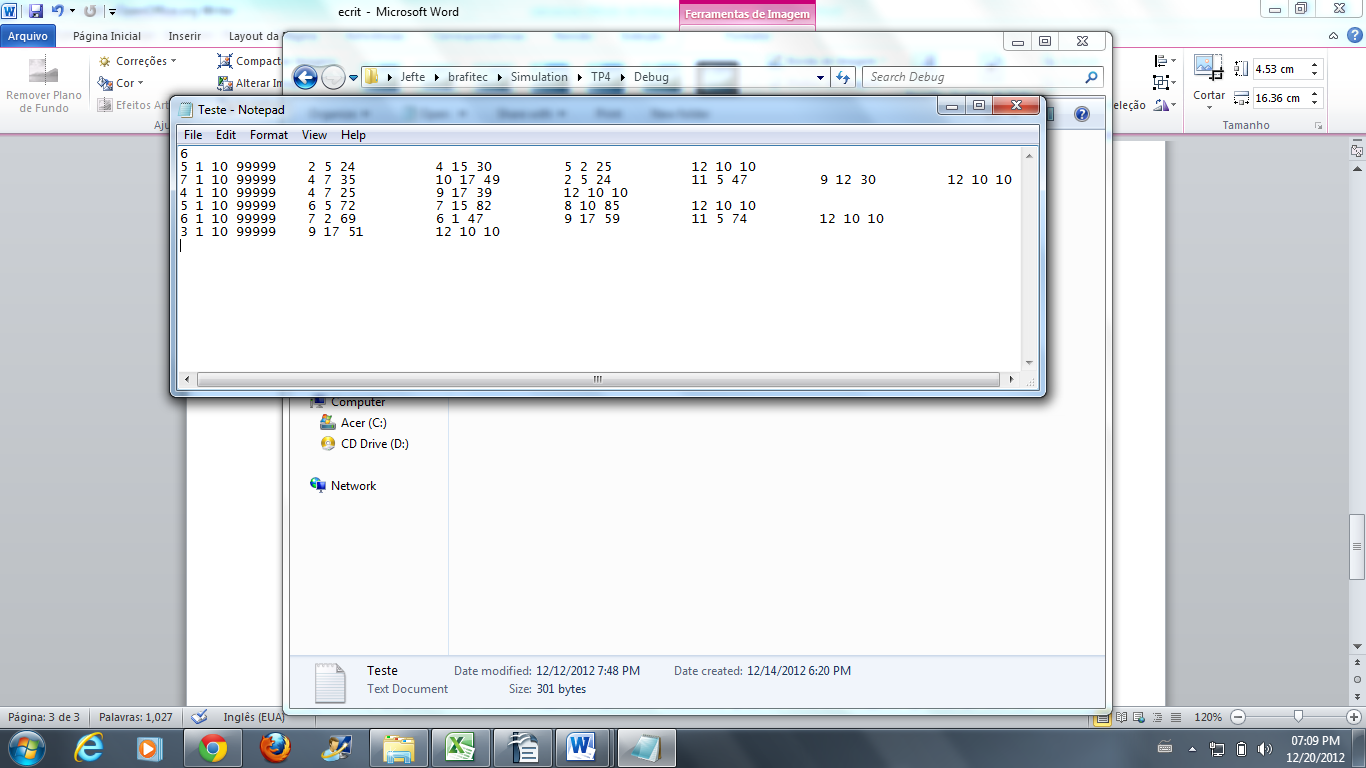


figure 1 – fichier de lecture

Le premier numéro est la quantité de gammes qui a dans le processus, alors dans chaque ligne il y a les informations pour faire la lectures de chaque gamme, à savoir : Le premier numéro de chaque ligne est la quantité d'opérations, après de trois en trois numéros on a les informations de chaque opération : Premier numéro de la cuve, deuxième et la durée minimale la pièce dans la cuve, et le dernier est la durée maximale.

Pour faire la lecture on a créé une fonctions qui s'appelle **void Lire\_Fichier(T\_Liste\_Gamme &Gammes, string &nom),** que va recevoir comme paramètres la liste avec gammes en Gammes et le nom du fichier en nom. Cette fonction va ouvrir le fichier comme lecture et ensuite elle va commencer a faire la lecture.

On a passé les paramètres pour références alors les valeurs qui étaient changé de la Gamme vont continuer changé après le fin de la fonction.

## Simuler()

La fonction **void Simuler(RichTextBox^ text, int \_Qt\_Max\_Piece\_Syst, int \_type, int Qt\_Pieces, int \*\_ordre, int \_Type\_Bloq, int \_type\_Retard)** a nombreuses utilisations et elle est la fonction du départ du programme. Elle reçoit comme paramètres d'entrée les options sélectionnées par l'utilisateur sur l'écran d'accueil et qui seront utilisés pour définir le fonctionnement du système. Après il y a une photo du programme avec tous le options d’utilisateur.

Cette fonctions initialise les structures avec les fonctions: Inicialise\_Gamme (), Inicialise\_VecteurRetard (),Inicialise\_Solution (); et après la lecture du fichier est effectuée en utilisant la fonction Lire\_Fichier().

Alors, Il y a un **Pour (ite = 0; ite<QtInterMax; ite++)**, où *QtInterMax* est egal à 1000 et représente le nombre maximum d'itérations. Quand le système est du type (1) ou (2) le *QtInterMax* = 1, et ainsi seulement une itération est effectuée.

Si le système est du type (2) et (3) un ordre d'entrée aléatoire est généré par la fonction **Generer\_ordre ().** Dans chaque itération, la fonction **Rouler\_Systeme ()** est appelée. Si Il y a un retard et le système n'accepte pas les retards, la fonction **Rouler\_Systeme ()** est encore appelé jusqu'à le délai n'existe plus.

Ensuite, la fonction **Mettre\_Dans\_Solution()** est appelée pour vérifier si la nouvelle solution obtenue dans cette itération est meilleure que la solution précédente. Si c'est le cas, la nouvelle solution remplace la précédente.

Si le système est du type (3)  la l’ordre sera modifié en utilisant la fonction **Chager\_Ordre()**, qui échange deux positions du vecteur *ordre.*

### Rouler\_Systeme()

Cette fonctions doit être appelle toute la fois que on est besoin de évaluer un vecteur ordre. Elle est une fonctions très importante, puis elle va définir le temps finale du processus (makespan) et elle remplira la fille sortie avec toutes les pièces du systèrme. Dans notre programme cette fonctions est do tipo void et reçoit comme paramètres les structures : T\_ordre, T\_Liste\_Gamme, T\_File, T\_Cuve et une vecteur de double. Au fin de la fonction elle retourne comme double le temps du processus.

Au début, la fonction effectue les initialisations nécessaires en utilisant les fonctions: **inicialise\_File(), Inicialiser\_Robot()** et **Inicialiser\_Cuves ()**.

S'il reste des pièces pour passer par le système, si la quantité de pièces dans le système n'est pas le maximum et si la cuve 1 est vide, la fonction **void Faire\_Entree\_Nouvelle\_Piece** *(const T\_Liste\_Gamme &Gammes,const T\_Ordre &Ordre,T\_Cuve \*Cuves, double &DS, int &Piece\_dans\_Syst, int &Qt\_piece\_Util, T\_Robot &Robot)* est appelée. Cette fonction crée une nouvelle pièce, conformément à l'ordre d'entrée, qui est placée dans la cuve 1. Il faut observer que si le utilisateur a choisi une système sans blocage ici la fonction Faire\_Entree\_Nouvelle\_Piece () ne va que être appelé si la prochaine pièce de l'ordre ne va pas passer pour les cuves, sauf la cuve 12, des pièces que sont dans le système. Pour clarifier, si la pièce 3 est dans le système et dans la cuve 4 mais elle doit encore passer pour les cuves 9 et 12 et dans le système on a que la fille d'entrée il y comme prochaine pièce la pièce de numéro 5 que va passer dans les cuves 1, 7, 6, 9, 11 et 12 cette pièce ne peut pas entrer, puis elle va passer sur la cuve 9 également à la pièce 3 qui est dans le système, alors la pièce 5 va entres seulement quand la pièce 3 entrer dans la cuve 9.

Sinon, la fonction **void Prendre\_Piece\_et\_mettre\_dans\_Cuve***(T\_Cuve \*Cuve, T\_Robot &Robot, int &P\_Syst, T\_File &Sortie, int &aux)*sera appellée. Cette fonction vérifie si il ya une pièce dans la cuve 1 et si la cuve de destination est libre. Dans l'affirmative, la pièce dans la cuve 1 sera transportée à la cuve suivante. Sinon, la cuve priorité est calculé avec la fonction **int Cuve\_Priorite** *(T\_Cuve \*Cuve, const T\_Robot& Robot)* alors la pièce sera transportée à la cuve suivante.

Suive la codage simplifié de la fonctions Rouler\_Systeme() et après da fonction Prendre\_Piece\_et\_mettre\_dans\_Cuve().

Fonctions Rouler\_Systeme ()

double Rouler\_Systeme (const T\_Ordre &Ordre,T\_File &Sortie, T\_Cuve \*Cuves){

T\_Robot Robot;

Robot.DPE = 0 ;

int DS = 0 ;

int TampMax = 999999 ;

int Piece\_dans\_Syst = 0 ;

tant que (DS < TampMax){

if (Piece\_dans\_Syst < Qt\_Max\_Piece\_Syst)&& (Cuves[1].etat == 0))

Faire\_Entree\_Nouvelle\_Piece (Cuve, DS, Ordre, Robot);

else

Prendre\_Piece\_et\_mettre\_dans\_Cuve(Cuve, Robot, Sortie);

}//fin tant que

}//fin fonction

Fonction Prendre\_Piece\_et\_mettre\_dans\_Cuve() :

void Prendre\_Piece\_et\_mettre\_dans\_Cuve( T\_Cuve \*Cuves, T\_Robot &Robot, T\_File &Sortie){

int Cuve\_Destin = Cuve[1].Piece.Gamme.Liste\_Operation[2].cuve;

int CuvePrio; // est cuve avec priorite

si (Cuve[1].etat == 1)&&(Cuve[1]..d\_sortie\_min <= Robot.DPE)&&(Cuve[CuvDest].etat == 0){

Temps\_Moviment\_Robot (Robot, Cuve); // faire mouvement du robot

T\_Piece = Cuve[1].Piece;

Ajouter\_Piece\_dans\_Cuve(Cuve[CuvDest], Piece, Robot.DPE);

}

sinon {

CuvPrio = Cuve\_Priorite (Cuve,Robot);

Cuve\_Destin = Prochaine\_Cuve (CuvPrio, Cuve[CuvPrio].Piece);

Temps\_Moviment\_Robot (Robot, Cuve); // faire mouvement du robot

Ajouter\_Piece\_dans\_Cuve(Cuve[CuvDest], Piece, Robot.DPE);

}

}//fin de la fonction

Résutats

Pour générer les résultats on a utilisé une ordinateur : Acer; Model - AO722 ; Processeur  - AMD C-60 APU avec Radeon (tm) HD Graphics 1.00 GHz ; RAM – 3.73GB ; Système Type – 64 bit Operation System, System opérationnelle – Windows 7 Home Premium. Langage C++ en Visual Studio 2008.

## 1 – Montrer qu'en changeant l'ordre des pièces, on change le makespan :

Pour afficher cette proposition, on a choisir le système (1) qui est le système qui va générer les ordres de manière aléatoire, on a mis la quantité maximale de pièces dans le système (Nmax) comme 2, on mis la quantité de pièces comme 6 et on a choisi une système qui permit blocage et retard. Alors le tableau au-dessous affiche les résultats de temps du processus (makespan) :

|  |  |  |  |
| --- | --- | --- | --- |
| **seq** | **Ordre d' entrée** | **Makespan** | **retard Total** |
| 1 | 2 -4 - 5 - 6 - 3 - 1 | 316.75 | 0 |
| 2 | 3 - 5 - 6 - 4 - 1 - 2 | 330.25 | 0 |
| 3 | 4 - 1 - 3 - 5 - 6 - 2 | 330.25 | 0 |
| 4 | 2 -5- 1 - 6 - 3 - 4 | 337.25 | 6.5 |
| 5 | 4 -6 - 1 - 5 - 2 - 3 | 340.75 | 13 |
| 6 | 1 -2 - 3 - 4 - 5 - 6 | 349.75 | 13 |
| 7 | 1 - 4 - 5 - 3 - 6 - 2 | 352.25 | 0 |

On peut voir que presque tous les résultats ont la valeur de makespan différent. Ainsi on montre que l'ordre des pièces va changer le makespan. Comme on a choisir le système que permit retard on peut les observer dans la dernière column.

## 2 – Résultats avec algorithme de descente stochastique

On va faire une tableau avec les résultats. Pour faire ce tableau on a choisir la quantité de pièces et on a fait chaque valeur de pièce trois fois à obtenir les moyennes et les meilleurs valeurs. Alors le tableau qu'on a est le suivant :

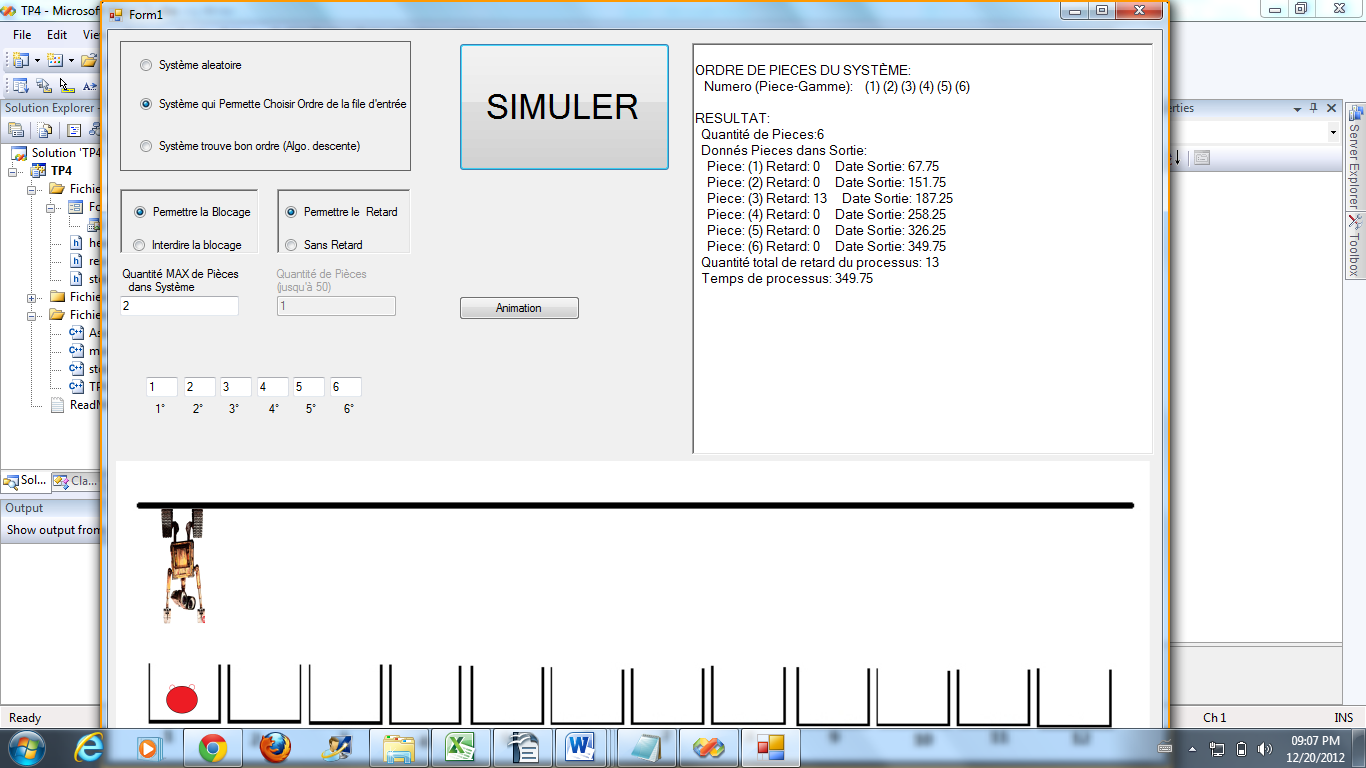
|  |  |  |  |
| --- | --- | --- | --- |
| **Quantité de Piéces** | **MAKESPAN** | | **Vecteur d’ordre du meilleur resultat** |
| **Moyenne** | **Meilleur** |
| 6 | 276.25 | 276.25 | (5) (6) (4) (3) (2) (1) |
| 12 | 541.75 | 539.25 | (5) (11) (1) (3) (10) (9) (2) (7) (12) (4) (6) (8) |
| 18 | 811.75 | 804.75 | (11) (17) (18) (14) (10) (3) (7) (12) (4) (15) (8) (13) (5) (16) (6) (2) (9) (1) |
| 24 | 1083.583 | 1076.75 | (11) (24) (1) (8) (16) (10) (18) (21) (23) (13) (6) (2) (5) (7) (15) (22) (3) (14) (4) (17) (12) (20) (9) (19) |
| 30 | 1362.25 | 1349.75 | (26) (22) (16) (30) (14) (5) (9) (10) (23) (12) (27) (7) (6) (4) (18) (21) (20) (25) (2) (29) (17) (11) (19) (15) (1) (24) (8) (28) (3) (13) |
| 36 | 1644.417 | 1631.75 | (20) (29) (19) (3) (34) (26) (25) (24) (8) (7) (12) (10) (2) (14) (22) (4) (15) (17) (13) (23) (35) (1) (33) (28) (9) (32) (31) (18) (36) (5) (30) (11) (6) (16) (21) (27) |
| 42 | 1939.083 | 1922.75 | (38) (9) (25) (15) (41) (5) (24) (8) (21) (19) (39) (31) (35) (23) (17) (40) (36) (6) (3) (13) (30) (26) (11) (28) (12) (18) (1) (27) (22) (2) (34) (16) (42) (32) (7) (20) (37) (33) (10) (14) (4) (29) |
| 48 | 2206.917 | 2187.25 | (24) (21) (22) (1) (11) (34) (28) (7) (47) (30) (32) (13) (42) (20) (23) (14) (45) (4) (15) (2) (37) (39) (19) (40) (35) (31) (12) (10) (27) (3) (9) (5) (29) (26) (46) (16) (25) (38) (43) (48) (33) (36) (17) (6) (41) (18) (44) (8 |

On peut observer que pour les petites instances de 6 pièces la moyenne est égal au meilleur résultats, ainsi on peut croire que le programme a trouvé le valeur optimale de cette instance.

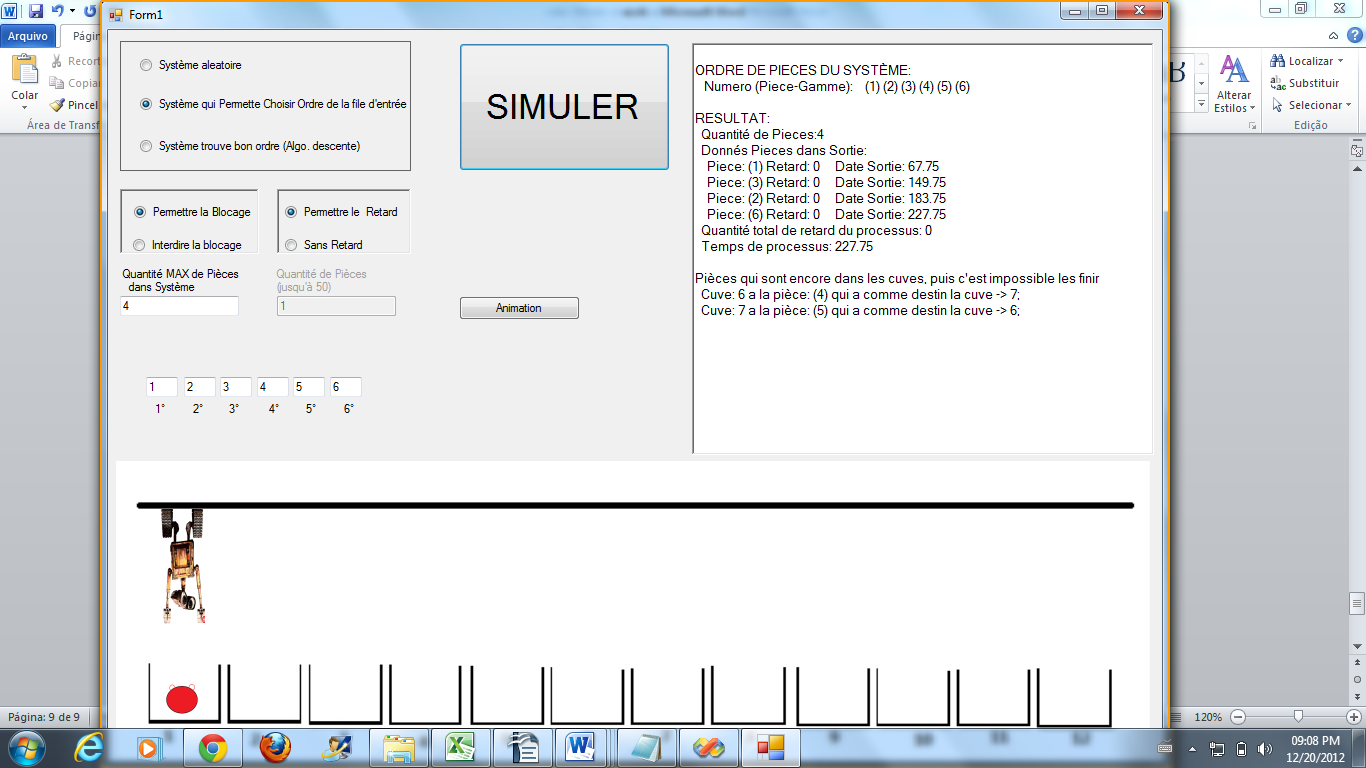
## 3 – Photos exécution du problème

1. Système qui permit blocage et retard

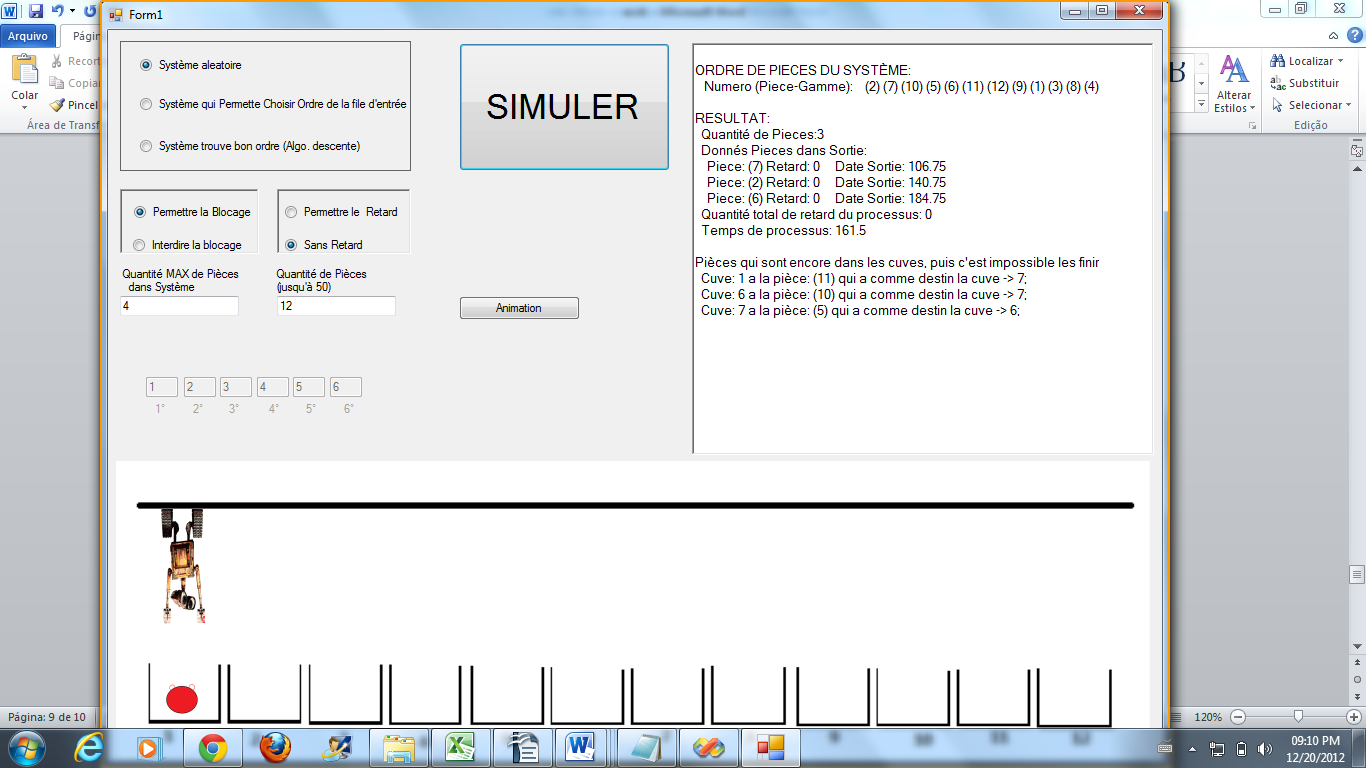
Avec Retard de 13 de la pièce 3 :



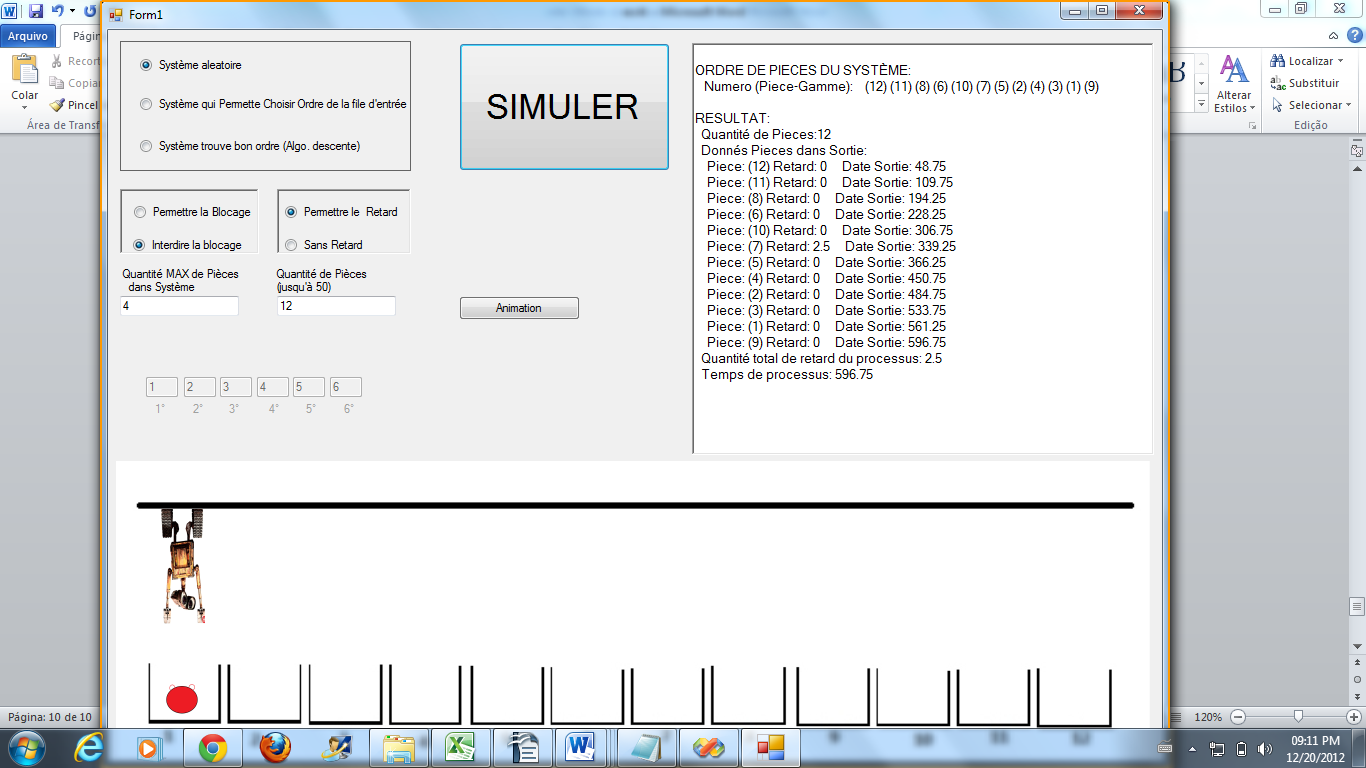
Avec Blocage de pièce 4 et 5 :



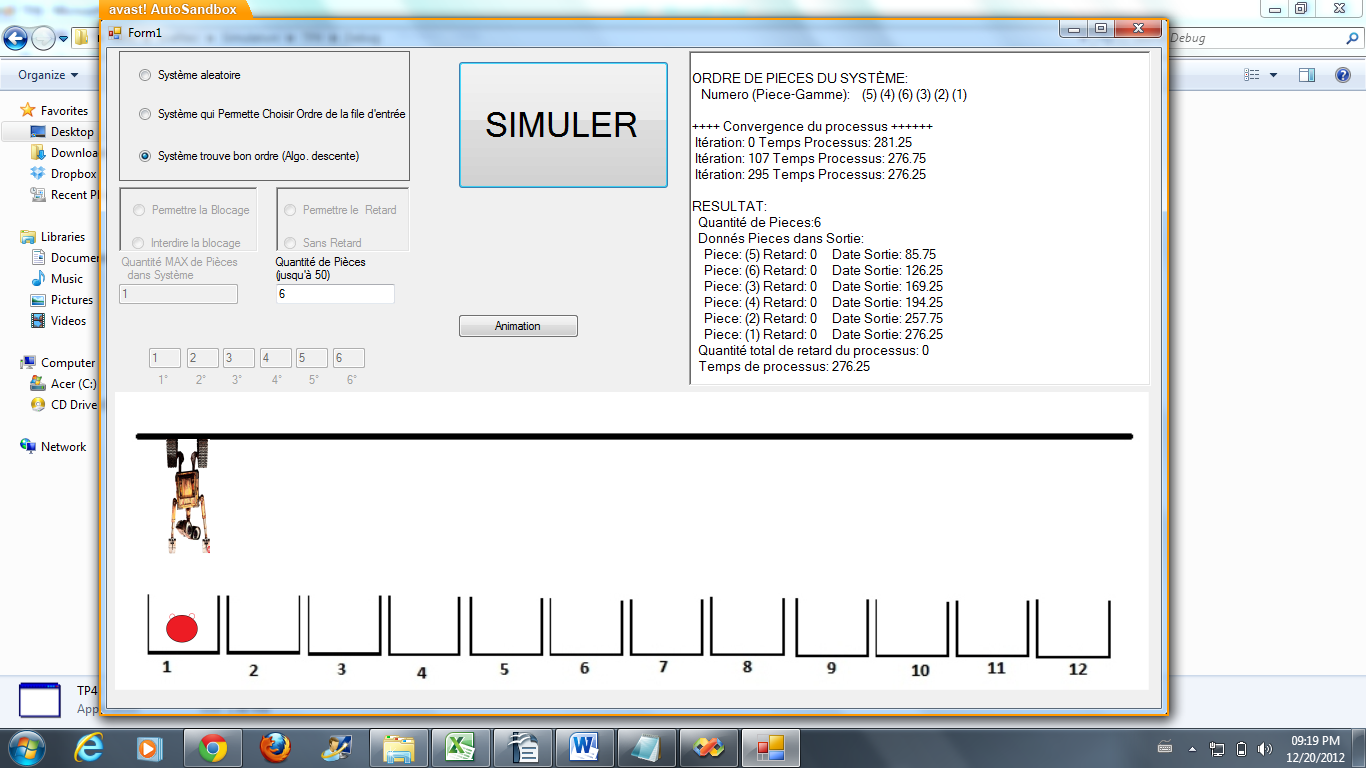
1. Système aleatoire qui permit la blocage mais interdire le retard



1. Système aléatoire qui interdire la blocage mais permettre le retard des pièces dans la cuve



1. Système qui trouve bon ordre (Photo complet du programme) :



Conclusion

Dans ce travail pratique on a créé comme nouvelle une animation graphique. Le programme est un peut lent quand le numéro de pièces est supérieur que 30 et dans la plupart des cases la options de interdire le retard marche très bien mais sur les cases que la première pièce est la pièce que cause le retard on n’obtiens pas le résultats espérer, puis même se on retard la pièce, elle jamais va passer pour le traitement du robot sans retard.

Bibliographie

[1] Laettia JOURDAN, 2003, Thèse de : Métaheuristique pour l'extraction de connaissances : Application à la Génomique.