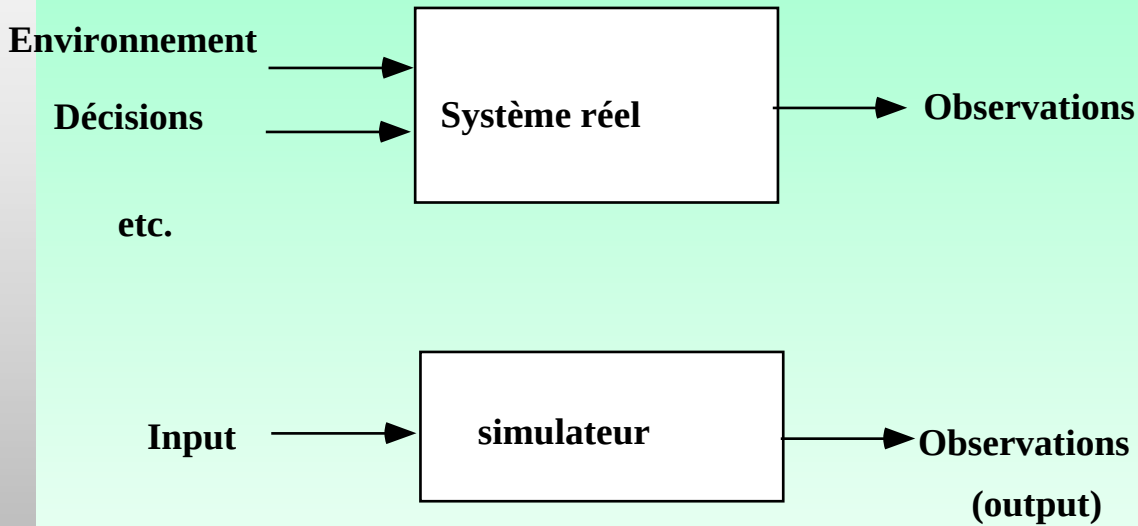
Principes de base des modèles de simulation

INTRODUCTION

- La simulation est une technique de résolution de problème.
- ORIGINE: Statistiques, analyse de systèmes complexes, probabilités, modélisation.
- **ALGORITHMIQUE** (en général)
  - ♦ Problème bien défini
  - ♦ Choix des techiques: Évident
  - ♦ Objectif visé est clair

  SIMULATION
  - ♦ Objectifs moins bien définis
  - ♦ Problèmes complexes où les techniques standards sont inutilisables.
- L'ESSOR DE LA SIMULATION EST DÛ PRINCIPALEMENT À L'ORDINATEUR.

INTRODUCTION



- POURQUOI SIMULER?

Permet d'étudier un système, de faire des expériences, sans avoir à assumer les coûts véritables de nos erreurs, de prédire le comportement d'un système.

EXEMPLES

EXEMPLES

- Jeux vidéos: Simulateur de vol, alunissage, combats, etc.

- Outils d'aide à la décision:

  - STRATÉGIES DE PRODUCTION
  - CHOIX D'UNE CONCEPTION
  - STRATÉGIES DE MARKETING
  - ENTREPRISES DE SERVICE
  - etc.

- Validation d'une théorie (chimie, physique, biologie, etc.):

Avantages et inconvénients de la simulation

- Permet de s'assurer qu'un système répond aux besoins.
  - Il y a des milliers d'applications possibles.
  - C'est non-destructif.
  - On peut reprendre une expérience autant de fois que l'on veut, dans les mêmes conditions.
  - PERMET D'ÉTUDIER DES SYSTÈMES TRÈS COMPLEXES.

  NOTE: Malheureusement, les temps de calculs en général sont très élevés.

  Plusieurs études de simulation sont faites avec une approche trop simpliste $\Rightarrow$ CONCLUSIONS ERRONÉES

Définition d'un système

Systèmes ≡ Assemblage de personnes ou d'objets réunis par des interactions.

Ensemble d'éléments qui agissent et interagissent dans un objectif commun.

Relations existant entre ces éléments.

EXEMPLES:

- cellule
- être vivant
- écosystème
- ville
- système d'inventaire
- transport routier
- etc.
- organe biologique
- espèce animale
- société commerciale
- économie d'un pays

Définition d'un système

Dans tout système, il y a cette difficulté de l'isoler de son milieu,

une infinité d'éléments qui interviennent,
une infinité de caractéristiques qui interviennent,
une infinité d'interactions qui interviennent,
plusieurs facteurs d'incertitude.

Il s'agit alors:

- Comprendre les systèmes
- Les analyser
- Mettre au point de nouveaux systèmes,
  modifier des systèmes existants, améliorer le fonctionnement des systèmes.

Représentation d'un modèle

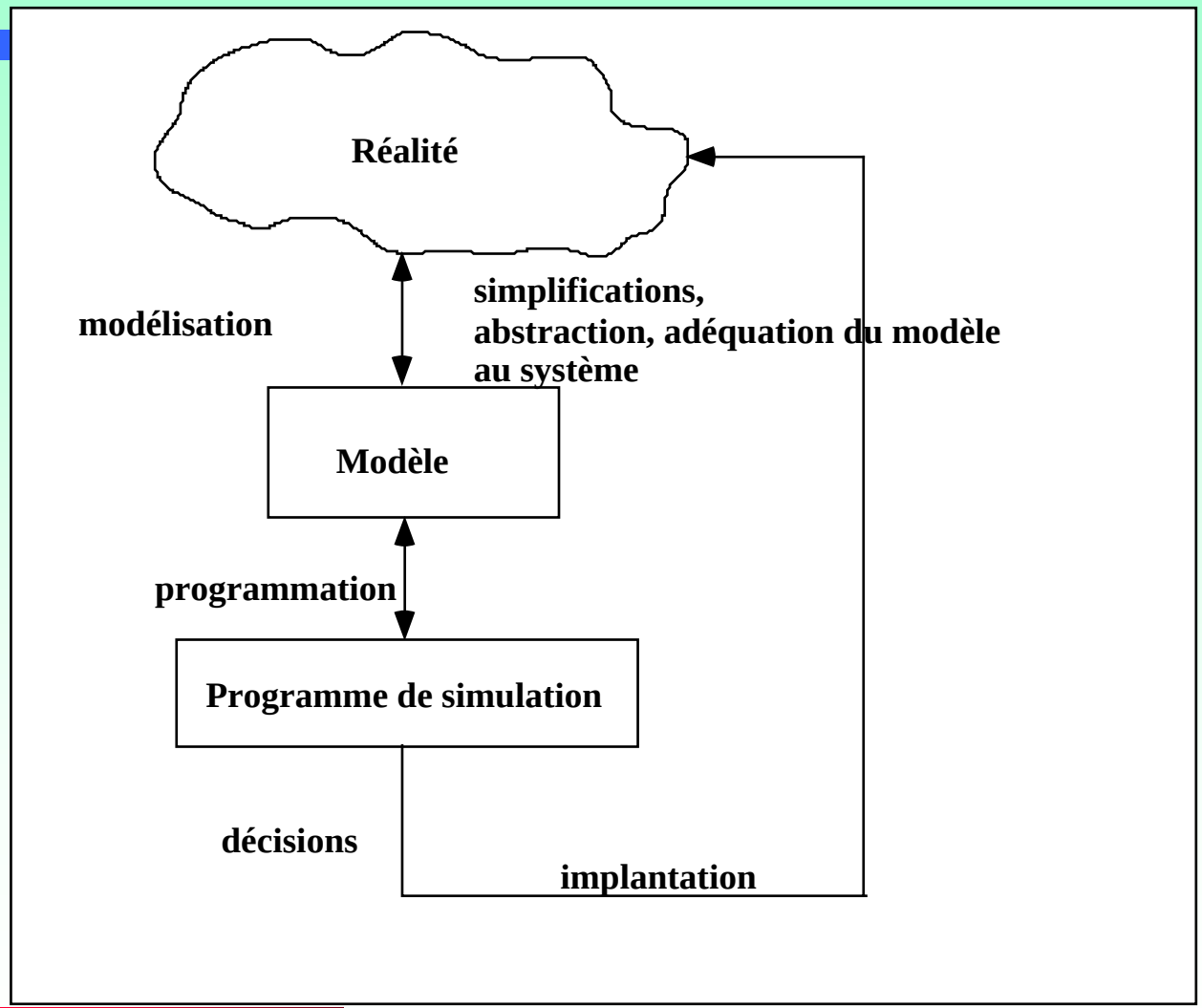
PHYSIQUE: maquette chimique
bassin hydraulique
soufflerie aérodynamique
ailes d'avion, bateaux, barrages
circuits électriques, etc.

ANALOGIQUE: (on peut remplacer ce système par un montage électrique)

équations différentielles

SYMBOLIQUE: algèbre,
calcul différentiel et intégral
recherche opérationnelle,
programmes, etc.

Modélisation



Déroulement d'une étude impliquant la simulation (ou la modélisation en général)

PROCESSUS DÉFINITION DU PROBLÈME;

- Discussion avec les intéressés;
- Formulation du problème et des objectifs de l'étude;
- Examen des techniques de solution;
- Cueillette des statistiques;
- Définition du système à modéliser et des objectifs.

PROCESSUS MODÉLISATION

- Formulation du modèle, documentation, validation;
- Programmation, vérification et validation;
- Choix d'un plan d'expérience;
- Expérimentation, obtention et analyse des résultats.

PROCESSUS IMPLANTATION;

- Documentation, production d'un rapport;
- Utilisation du simulateur comme outil d'aide à la décision.

Déroulement d'une étude impliquant la simulation (ou la modélisation en général)

BEGIN

Répéter

Définition du problème;

Modélisation;

jusqu'à ce que le simulateur soit satisfaisant;

Implantation;

END.

OBJECTIF: - Formuler un modèle qui renferme suffisamment de détails et qui colle suffisamment à la réalité de sorte que les hypothèses faites lors de la construction du modèle peuvent être déduites du système.

Notes:

- Commencer par le plus simple, détailler ensuite si nécessaire.
- Un modèle simple et réaliste laisse voir que l'on a vraiment compris ce qui est important.
- Plusieurs voient la simulation comme un outil de dernier recours.

Déroulement d'une étude impliquant la simulation (ou la modélisation en général)

Notes:

- La simulation n'a de sens que si on a les données pour construire un modèle (et estimer les paramètres) de façon assez précise et réaliste.

$\Rightarrow$ $$

- ÉQUILIBRE : RÉALISME DU MODÈLE
  vs
  ANALYSE STATISTIQUE.

- FLEXIBILITÉ: En pratique, on ne cesse jamais de modifier les modèles et les programmes.

- En général, la simulation permet d'évaluer ou d'estimer, mais rarement ou difficilement d'optimiser.

Approximations dans les modèles

- Approximations fonctionnelles (linéaire, quadratique,...)
  Habituellement, elles sont valides seulement dans un certain voisinage.

- LOIS DE PROBABILITÉS SIMPLIFIÉES.
- Variables aléatoires indépendantes.
- Agrégation: temps, objets, ressources,...

- Stationnarité:
  En pratique, les systèmes sont rarement stationnaires.
  Un état stationnaire (horizon infini) est utile pour approximer une solution ou donner une formulation analytique).

- Valider avec des données différentes de celles utilisées pour construire le modèle.
- SE MÉFIER DE L'APPROCHE "TOP-DOWN".

Exemple I - Une file d'attente

RÉALITÉ: UN QUAI DE CHARGEMENT DE MARCHANDISES
où
1 seul bateau est chargé à la fois.

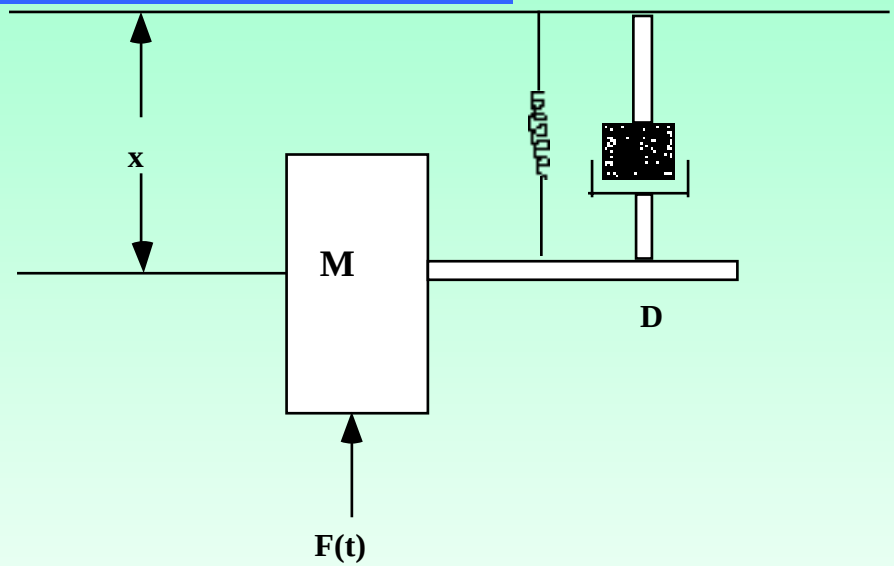
MODÈLE: File d'attente M/M/1
(arrivée et durée de service exponentielles, 1 serveur)

HYPOTHÈSES:

- Les bateaux arrivent selon un processus de Poisson de paramètre λ.
- Le temps de service d'un bateau est une v.a. exponentielle de paramètre ω.
- Ordre de priorité: ordre d'arrivée.
- La zone d'attente a une capacité infinie.

Question posée: Estimer le temps moyen d'attente des bateaux dans la file.

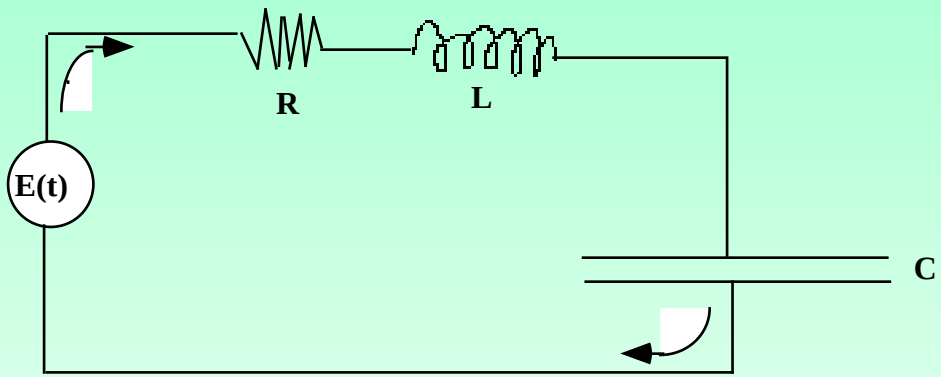
Exemple II - Suspension d'une automobile



M	masse de la roue
D	coefficient d'absorption de l'amortisseur
k	coefficient d'élasticité du ressort

$$M\,x'' + D\,x' + k\,x = k\,F(t)$$

Exemple III - Montage électrique (Représentation analogique)



E(t): source de courant de voltage variable
R: résistance
L: inductance
C: capacitance
q: quantité de courant

$$L'' + Rq' + q/C = E(t)/C$$

Exemple IV - Méthode analytique (suspension d'une automobile)

On peut aussi écrire: $x'' + 2\zeta\,\omega\,x' + \omega^2 x = \omega^2 F(t)$

où $2\zeta\,\omega = D/M$ et $\omega^2 = k/M$,

ζ : facteur d'amortissement

Il n'y a pas d'oscillation dans le système ssi l'éq ^{n} caractéristique associée possède des solutions réelles.

L'éq ^{n} caractéristique associée a la forme: $x^2 + 2\zeta\omega\,x + \omega^2 = 0$.

Cette équation aura des racines réelles ssi $4\zeta^2\omega^2 - 4\omega^2 \geq 0$ i.e. $\zeta^2 - 1 \geq 0$.

Il faut donc que $\zeta \geq 1$ pour qu'il n'y ait pas oscillation. Puisque $\zeta = D/(2M\omega)$ et $\zeta \geq 1$, il n'y a pas d'oscillations ssi $D^2 \geq 4Mk$ (correspond à un amortissement brusque)

$\therefore$ D^2 doit être près de (<) $4Mk$ pour que les oscillations aillent en s'amenuisant de façon correcte.

Autres définitions

Entités : Objets faisant partie du système, ou évoluant dans le système (quai, bateau, etc.)

Attribut: Caractéristique d'une entité (une place de chargement)

État du système: Les valeurs d'un ensemble de variables qui suffisent à décrire le système à un instant donné.

exemple: nombre de bateaux dans le système,
instant d'arrivée de chacun de ces bateaux,
etc.

Caractérisé par l'ensemble des entités à ce moment,
par les valeurs des attributs à ce moment,
par les activités qui se déroulent à ce moment.

Espace d'états: Ensemble de tous les états possibles du système.

SYSTÈME À ÉTAT DISCRET ("DISCRETE STATE")

lorsque l'espace d'états est fini ou dénombrable. Ex.: 1 quai de chargement.

Autres définitions

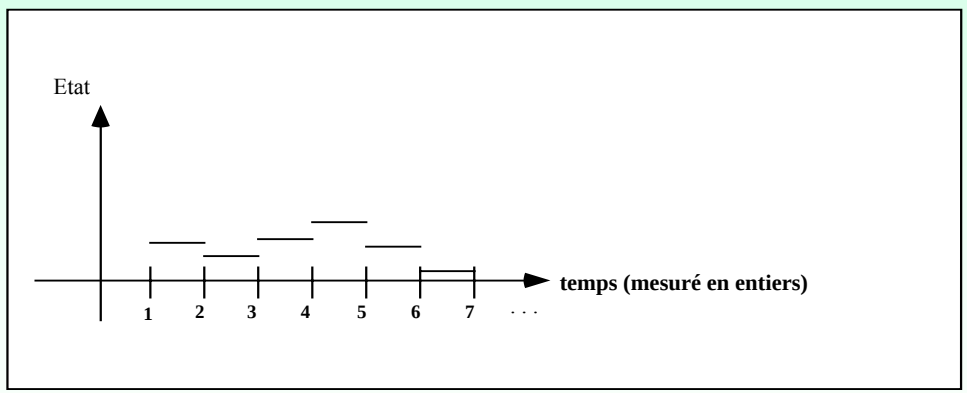
SYSTÈME À ÉTAT CONTINU: ("CONTINUOUS STATE")

ESPACE D'ÉTATS CONTINU .

MODÈLE EN TEMPS DISCRET ("DISCRETE TIME")

Lorsque l'état du système ne peut changer qu'en des instants fixés d'avance, régulièrement espacés dans le temps.

Exemple :



Autres définitions

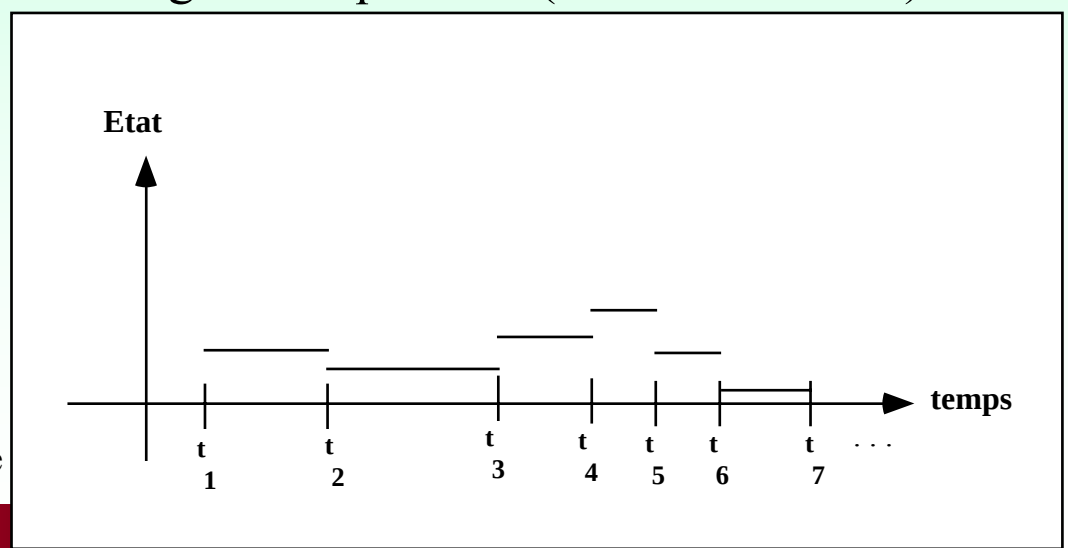
MODÈLES À ÉVÉNEMENTS DISCRETS (“DISCRETE EVENT”)

Le temps évolue continûment, mais l’état du système ne “change” que par sauts, à un nombre dénombrable d’instants. Ces instants sont des points dans le temps où se produisent **des événements**.

ÉVÉNEMENT (“EVENT”)

Un fait qui se produit, qui peut changer brusquement (instantannément) l’état du système.

Ex.: arrivée ou départ d’un bateau.



Autres définitions

ACTIVITÉ : Se déroule à travers le temps.
Exemple: Chargement d'un bateau.

Événements :
- exogènes: se situe à l'extérieur du modèle
- endogènes: se situe à l'intérieur du modèle.

Modèle :
- fermé: aucun événement exogène et les entités demeurent à l'intérieur du système
- ouvert: sinon

Modèle à rétro-action: certaines entités sont susceptibles de repasser plusieurs fois dans le système.

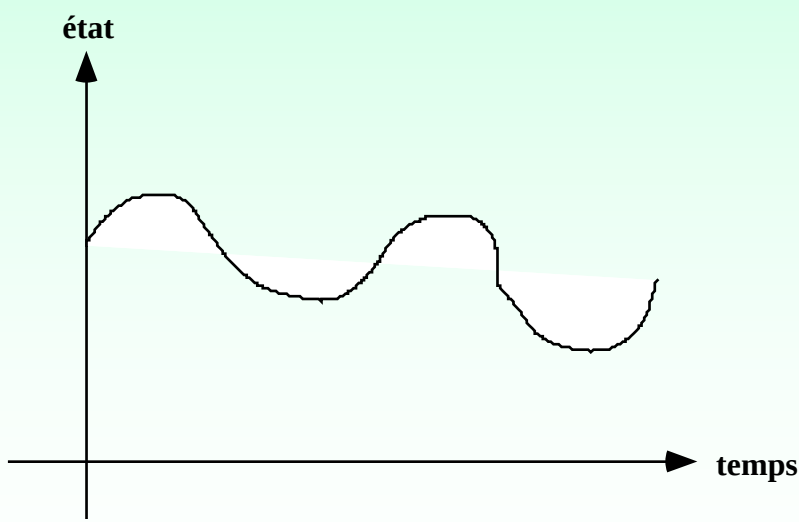
Autres définitions

Modèle à évolution continue (simulation continue)

Le temps avance continûment et l'état peut aussi changer "doucement" de façon continue.

Exemple:

- objet qui se déplace
- bateau qui se décharge
- corps qui se refroidit
- piscine qui se vide
- etc.



L'évolution est habituellement régie par des équations différentielles.

Autres définitions

Modèles hybrides : évolution continue + événements discrets

Système :

- Statique: On étudie le système à un instant fixé.
  Ex.: Montage fixe: rupture ou non d'un pont
- Dynamique: On s'intéresse à l'évolution du système dans le temps.

Système :

- Déterministe: Rien d'aléatoire, on peut savoir d'avance tout ce qui va se passer.
- Stochastique: Contient des éléments aléatoires, ce qui va se passer dépend des valeurs que vont prendre certaines v. a. (instants d'arrivées, durées de service, ...)

But du cours

Dans ce cours, on s'intéressera surtout
à la simulation des systèmes
dynamiques,
stochastiques
à
événements discrets

Exemple : Simulation d'un quai de chargement (File d'attente M/M/1)

ÉTAT: LISTE DES BATEAUX DANS LE SYSTÈME,
LEUR INSTANT D'ARRIVÉE,
LEUR DURÉE DE SERVICE PRÉVUE.
LE BATEAU EN SERVICE, (S'IL Y A LIEU), LE STATUT DU SERVEUR,
LE TEMPS QU`IL LUI RESTE POUR COMPLÉTER SON SERVICE.
L'INSTANT COURANT DE LA SIMULATION: L'HORLOGE.

ÉVÉNEMENTS:

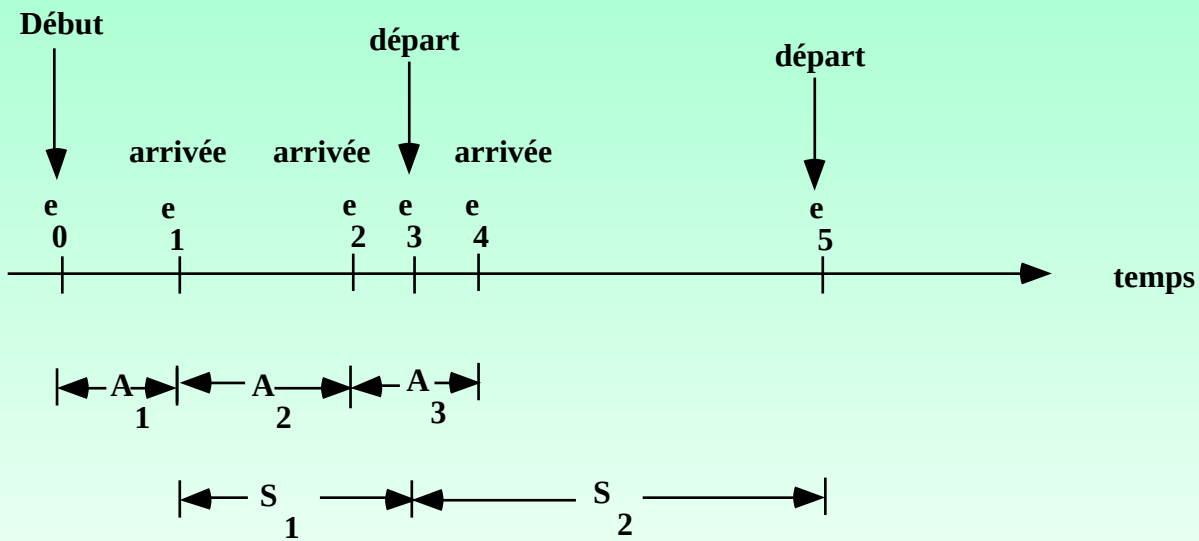
ARRIVÉE D'UN BATEAU,
FIN D'UN SERVICE.

COMPTEURS: (SERVENT À RECUEILLIR LES STATISTIQUES)

N_ACC : # de bateaux ayant accosté (i.e. ayant fini d'attendre)
ATT : Attente totale des bateaux ayant accosté.
ATT/N_ACC: À la fin, l'attente moyenne par bateau.

Fin de la simulation: Arrêt lorsque N_ACC = N (une valeur fixée d'avance).

Exemple : Simulation d'un quai de chargement (File d'attente M/M/1)

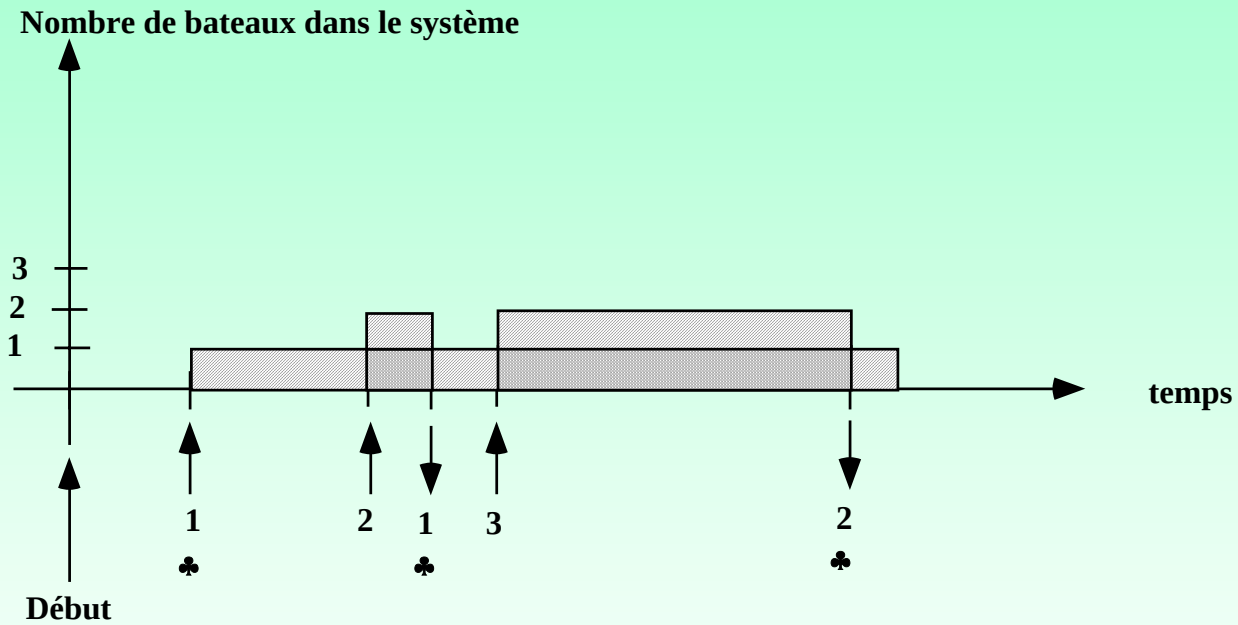


$e_0, e_1, ..., e_5$: les événements

Les variables aléatoires (I.I.D.) pour lesquelles il faut générer des valeurs sont:

- les temps inter-arrivées $A_1, A_2, A_3, ...$
- les durées de service $S_1, S_2, ...$

Exemple : Simulation d'un quai de chargement (File d'attente M/M/1)



N_ACC compte le nombre de ♣

ATT calcule la surface doublement hachurée

Organisation d'un simulateur

Variables pour:

- mémoriser l'état du système
- compteurs statistiques

Liste des événements futurs prévus:

- Prochaine arrivée
- Prochain départ

Routines pour:

- initialiser une simulation
- gérer l'évolution
- réaliser chaque événement
- générer des valeurs "aléatoires"
- imprimer un rapport.

Construction d'un simulateur en C++
Quai_de_chargement

```
/**************************************************************************

Simulation d'un quai de chargement modélisé par une file d'attente M/M/1.

Le système est initialement vide et on simule jusqu'à ce que le N-ième bateau commence
son service (i.e. accoste au quai).

**************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

Construction d'un simulateur en C++
Quai_de_chargement

```
const double Maximum_des_reels = 3.4E38;
enum Bool {false, true};

long souche;

enum Evenement {Arrivee, Depart};          // Types d'événement
int N;                                     // Nombre d'accostage à observer
float temps_moyen_entre_2arrivees;         // Temps moyen entre 2 arrivées
float duree_moyenne_chargement;            // Durée moyenne d'un chargement
float Temps;                               // Instant courant de la simulation
enum Bool Quai_Libre;                      // Indique si le quai est libre
```

Construction d'un simulateur en C++
Quai_de_chargement

```
struct Description_Bateau
{
        float Instant_Arrivee_Bateau;        // Instant d'arrivée de ce bateau
        float Duree_Service;                 // Durée de service pour ce bateau
        struct Description_Bateau * Bateau_Suivant;
                                             // Le bateau suivant dans la file
};

enum Evenement Evenement_courant;            // Événement que nous sommes en train
                                             // de traiter.
float Instant_Prochain_Evenement[2];
                                             // Instant d'occurrence du prochain
                                             // événement prévu de chaque type
```

Construction d'un simulateur en C++
Quai_de_chargement

```
struct Description_Bateau*Prochain_Bateau; // Premier bateau arrivé dans la file
struct Description_Bateau* Dernier_Bateau; // Dernier bateau arrivé dans la file
int Nombre_de_bateaux_accostes;            // Nombre de bateaux ayant accosté
float Temps_total_attente_bateaux_accostes; // Attente totale des bateaux ayant accosté
void main()
{
        float Duree_entre_deux_arrivees();
        float Duree_du_service();
        void Lire_Donnees();
        void Initialisation_Simulation();
        void Arrivee_bateau();
        void Depart_bateau();
        void Rapport();
        Lire_Donnees();
        Initialisation_Simulation();
```

Construction d'un simulateur en C++
Quai_de_chargement

```
while (Nombre_de_bateaux_accostes < N)
{
        if(Instant_Prochain_Evenement[Arrivee] <
                                        Instant_Prochain_Evenement[Depart])
                Evenement_courant = Arrivee; else Evenement_courant = Depart;
        Temps = Instant_Prochain_Evenement[Evenement_courant];
        switch (Evenement_courant)
        {
                case Arrivee    : Arrivee_bateau(); break;
                case Depart     :  Depart_bateau(); break;
        };
};
Rapport();
}
```

Construction d'un simulateur en C++
Quai_de_chargement

```
float Duree_entre_deux_arrivees()
{
        // Génère la durée entre deux arrivées successives.

        return (float) (-log(1.0 - (float)((rand() % 10000) / 10000.0))
                * temps_moyen_entre_2arrivees);
}

float Duree_du_service()
{
        // Génère une durée de service.

        return (float) (-log(1.0 - (float)((rand() % 10000) / 10000.0))
                * duree_moyenne_chargement);
}
```

Construction d'un simulateur en C++
Quai_de_chargement

```
void Lire_Donnees()
{
        /*      Lecture du nombre de bateaux à accoster, du temps moyen entre deux
                arrivées de bateaux et de la durée moyenne d'un chargement.
                Impression des données d'entrée.                                */

        printf("  Veuillez fournir les renseignements suivants : \n\n");
        printf("  Nombre de bateaux a accoster = ");
        scanf("%d", &N);
        printf("  Temps moyen entre deux arrivees de bateaux = ");
        scanf("%f", &temps_moyen_entre_2arrivees);
        printf("  Duree moyenne d'un chargement = ");
        scanf("%f", &duree_moyenne_chargement);
        printf("  ------------------------------\n");
}
```

Construction d'un simulateur en C++
Quai_de_chargement

```
void Initialisation_Simulation()
{

/*      Initialise le système à vide, initialise l'horloge et tous les compteurs à 0,
        le premier événement est une arrivée et on prévoit son instant d'occurrence */
        Quai_Libre = true;
        Prochain_Bateau = NULL;
        Temps = 0.0;

        souche = time(NULL);      srand((int)souche);
        Temps_total_attente_bateaux_accostes = 0.0;
        Nombre_de_bateaux_accostes = 0;
        Instant_Prochain_Evenement[Arrivee] = Duree_entre_deux_arrivees();
        Instant_Prochain_Evenement[Depart] = (float) Maximum_des_reels;
}
```

Construction d'un simulateur en C++
Quai_de_chargement

```
void Arrivee_bateau()
{
        // Exécution de l'événement : arrivée d'un bateau.

        struct Description_Bateau * Bateau;
        if (Quai_Libre == true)                 // Ce bateau n'attend pas.
        {
                Nombre_de_bateaux_accostes += 1;
                Quai_Libre = false;
                Instant_Prochain_Evenement[Depart] = Temps + Duree_du_service();
        }
```

Construction d'un simulateur en C++
Quai_de_chargement

```
else        // Le bateau entre dans la file d'attente.
{
        Bateau = (struct Description_Bateau *) malloc(sizeof(struct Description_Bateau));
        Bateau -> Instant_Arrivee_Bateau = Temps;
        Bateau -> Duree_Service = Duree_du_service();
        Bateau -> Bateau_Suivant = NULL;
        if(Prochain_Bateau == NULL)                    // La file était vide.
        {
                Prochain_Bateau = Bateau;    Dernier_Bateau  = Bateau;
        }
        else
        {
                Dernier_Bateau -> Bateau_Suivant = Bateau; Dernier_Bateau = Bateau;
        }
};
Instant_Prochain_Evenement[Arrivee] = Temps + Duree_entre_deux_arrivees();
}
```

Construction d'un simulateur en C++
Quai_de_chargement

```
void Depart_bateau()          // Exécution de l'événement : départ d'un bateau.
{
        struct Description_Bateau * Bateau;

        if(Prochain_Bateau  == NULL)       // Le système se vide.
        {
                Quai_Libre = true;
                Instant_Prochain_Evenement[Depart] = (float) Maximum_des_reels;
        }
```

Construction d'un simulateur en C++
Quai_de_chargement

```
        else      // Le prochain bateau accoste.
        {

                  Bateau = Prochain_Bateau; Nombre_de_bateaux_accostes += 1;
                  Temps_total_attente_bateaux_accostes +=
                                        Temps - (Bateau -> Instant_Arrivee_Bateau);
                  Instant_Prochain_Evenement[Depart] = Temps +
                                                          Bateau ->Duree_Service;
                  Prochain_Bateau = Bateau -> Bateau_Suivant;
                  free(Bateau);

        }
}
```

Construction d'un simulateur en C++
Quai_de_chargement

```
void Rapport()
{
        //      Imprime la durée de la simulation et l'attente moyenne par bateau.

        printf("  Durée de la simulation : %.3f\n", Temps);
        printf("  Attente moyenne par bateau : %.3f\n",
                Temps_total_attente_bateaux_accostes / N);
}
```

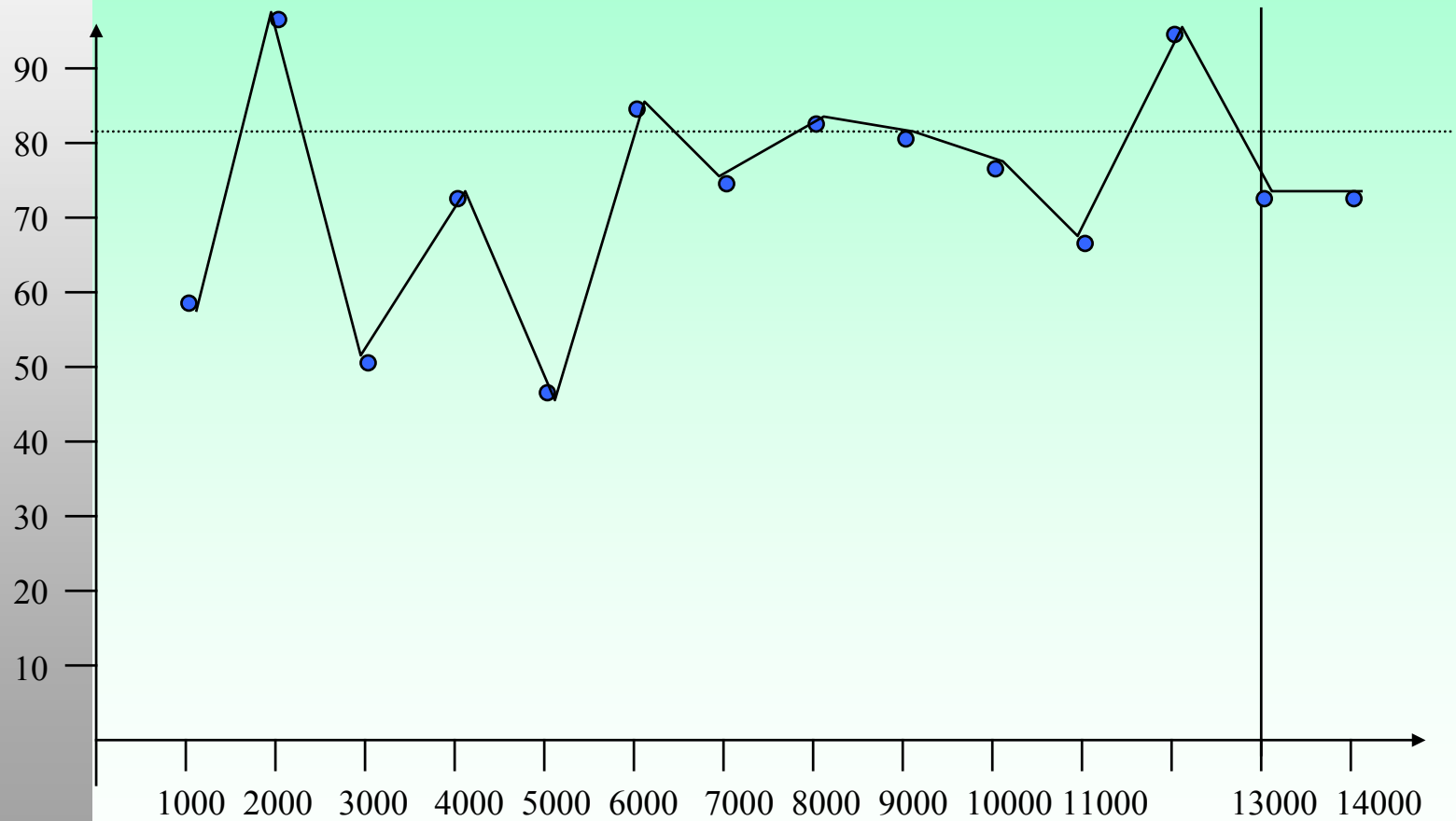
FIN

Analyse des résultats Quai_de_chargement

	ESSAI 1	ESSAI 2	ESSAI 3
Temps moyen inter-arrivées :	10	10	10
Durée moyenne d'un chargement :	9	9	9
Nombre de bateaux accostés :	40	1000	50 000
Durée de la simulation:	384.794	9330.586	467345.281
Attente moyenne par bateau :	37.292	58.930	82.757

À remarquer : La moyenne obtenue peut changer beaucoup en fonction du nombre d'accostages observés. Pourquoi?

Analyse des résultats Quai_de_chargement



Quai_de_chargement
Longueur moyenne de la file d'attente

- SUPPOSONS QUE L'ON VEUT AUSSI ESTIMER LA LONGUEUR MOYENNE DE LA FILE D'ATTENTE EN FONCTION DU TEMPS.

- On veut s'arrêter après T unités de temps, plutôt qu'à l'accostage du N^{ième} bateau.

- On peut prévoir un nouveau type d'événement, appelé "Fin_de_Simulation", qui a lieu à l'instant T.

- Temps_total_attente_bateaux : Attente totale de tous les bateaux durant [0,T] (dans la file).

  Temps_total_attente_bateaux_accostes contient déjà l'attente des bateaux ayant accosté; il suffit d'y ajouter l'attente de ceux qui sont encore dans la file d'attente.

- Temps_total_attente_bateaux / T: Longueur moyenne de la file d'attente.

Quai_de_chargement

Principaux changements dans le programme

A) dans les déclarations

```
enum Evenement {Arrivee, Depart, Fin_de_simulation};
float Duree_de_la_simulation;                    // Durée de la simulation
float Temps_total_attente_bateaux_accostes;
                                    // Attente totale des bateaux ayant accosté
float Temps_total_attente_bateaux;
                                    // Attente totale des bateaux dans la file
```

B) dans Lire_Donnees

```
        printf("  Duree de la simulation = ");
        scanf("%f", &Duree_de_la_simulation);
```

Quai_de_chargement

Principaux changements dans le programme

C) La fonction Rapport devient :

```
void Rapport()        // Imprime la durée de la simulation et l'attente moyenne par bateau.
{
        struct Description_Bateau * Bateau;
        printf("  Nombre de bateaux ayant accostes : %d\n",
                  Nombre_de_bateaux_accostes);
        printf("  Attente moyenne par bateau : %.3f\n",
                  Temps_total_attente_bateaux_accostes / Nombre_de_bateaux_accostes);
        Temps_total_attente_bateaux = Temps_total_attente_bateaux_accostes;
        Bateau = Prochain_Bateau;
        while (Bateau != NULL)
        {         Temps_total_attente_bateaux += Temps - Bateau -> Instant_Arrivee_Bateau;
                  Bateau = Bateau -> Bateau_Suivant;
        };
        printf("  Longueur moyenne de la file d'attente : %.3f\n",
                  Temps_total_attente_bateaux / Duree_de_la_simulation);
}
```

Quai_de_chargement
Principaux changements dans le programme

D) La principale fonction devient :

```
void main()
{
        float Duree_entre_deux_arrivees();
        float Duree_du_service();
        void Lire_Donnees();
        void Initialisation_Simulation();
        void Arrivee_bateau();
        void Depart_bateau();
        void Rapport();
        Lire_Donnees();
        Initialisation_Simulation();
```

Quai_de_chargement

Principaux changements dans le programme

```
do
{
        if(Instant_Prochain_Evenement[Arrivee] <
                                        Instant_Prochain_Evenement[Depart])
                Evenement_courant = Arrivee; else Evenement_courant = Depart;
        Temps = Instant_Prochain_Evenement[Evenement_courant];
        if(Temps >= Duree_de_la_simulation)
        {       Temps = Duree_de_la_simulation;
                Evenement_courant = Fin_de_simulation;
        };
        switch (Evenement_courant)
        {       case Arrivee          : Arrivee_bateau(); break;
                case Depart           :  Depart_bateau(); break;
                case Fin_de_simulation : Rapport(); break;
        };
} while (Evenement_courant != Fin_de_simulation);
}
```

Quai_de_chargement
Analyse des résultats

Période de simulation : [0, 1000]

	ESSAI 1	ESSAI 2	ESSAI 3
Temps moyen inter-arrivées :	10	10	10
Durée moyenne d'un chargement :	9	9	9
Nombre de bateaux ayant accosté :	88	105	106
Longueur moy. de la file d'attente:	1.341	3.137	5.787
Attente moy. par bateau :	14.672	29.520	53.884

On remarque que les résultats obtenus **varient beaucoup** d'une répétition à l'autre. La simulation ne fournit qu'une **estimation**, et non pas la vraie valeur de ce que l'on veut estimer. Dans ce cas-ci, la variance des estimateurs semble très grande.

Quai_de_chargement
Analyse des résultats

Période de simulation : [0, 50000]

	ESSAI 1	ESSAI 2	ESSAI 3
Temps moyen inter-arrivées :	10	10	10
Durée moyenne d'un chargement :	9	9	9
Nombre de bateaux ayant accosté :	5313	5327	5405
Longueur moy. de la file d'attente:	6.706	6.159	8.985
Attente moy. par bateau :	63.110	57.806	82.341

Ici, la durée de la simulation est beaucoup plus longue. Les estimateurs ont aussi une moins grande variance. À la limite, lorsque la durée de la simulation tend vers l'infini, l'attente moyenne par bateau et la longueur moyenne de la file d'attente tendent vers 81 et 8.1 respectivement. Ce sont les moyennes "théoriques", à l'état stationnaire (i.e. en moyenne sur une durée infinie).

Plus on simule longtemps, meilleur est l'estimateur, mais plus ça coûte cher!

Quai_de_chargement
Commentaires

- Dans cet exemple, peu d'événements peuvent être prévus en même temps (3 au max.)
  Dans certains cas, il peut y en avoir des centaines!

- On peut alors utiliser une liste (triée) des événements prévus.
  Les listes sont aussi utilisées à plusieurs autres fins.
  ex. liste des bateaux en attente
  " À venir "
- Les fonctions pour générer les valeurs "aléatoires" seront examinées plus tard.
  " À venir "
- Notre objectif était d'estimer des mesures de performance pour un système à l'état **stationnaire**. On commence avec un système vide. ⇒ Biais initial.

- **REMÈDES** : - simuler plus longtemps - réchauffement du simulateur.

Quai_de_chargement
Réchauffement du simulateur

```
/**************************************************************************
Simulation d'un quai de chargement modélisé par une file d'attente M/M/1.

Le système est initialement vide et on simule pendant une période [Duree_du_rechauffement,
Duree_de_la_simulation]. L'intervalle de réchauffement du simulateur est
[0, Duree_du_rechauffement].

**************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

Quai_de_chargement
Réchauffement du simulateur

```
const double Maximum_des_reels = 3.4E38;
long souche;
enum Evenement {Arrivee, Depart, Fin_du_rechauffement, Fin_de_simulation};
enum Bool {false, true};
float temps_moyen_entre_2arrivees;     // Temps moyen entre 2 arrivées
float duree_moyenne_chargement;        // Durée moyenne d'un chargement
float Temps;                           // Instant courant de la simulation
float Duree_de_la_simulation;          // Durée de la simulation
float Duree_du_rechauffement;          // Durée du réchauffement
enum Bool Quai_Libre;                  // Indique si le quai est libre
struct Description_Bateau
{
        float Instant_Arrivee_Bateau;// Instant d'arrivée de ce bateau
        float Duree_Service;                          // Durée de service pour ce bateau
        struct Description_Bateau * Bateau_Suivant;   // Le bateau suivant dans la file
};
```

Quai_de_chargement
Réchauffement du simulateur

```
enum Evenement Evenement_courant;  // Événement que nous sommes en train de traiter
float Instant_Prochain_Evenement[3];  // Instant d'occurrence du prochain événement prévu de
                                      // chaque type

struct Description_Bateau * Prochain_Bateau;      // Premier bateau arrivé dans la file
struct Description_Bateau * Dernier_Bateau;       // Dernier bateau arrivé dans la file
int Nombre_de_bateaux_accostes;                   // Nombre de bateaux ayant accosté
float Temps_total_attente_bateaux_accostes;       // Attente totale des bateaux ayant accosté
float Temps_total_attente_bateaux;                // Attente totale des bateaux dans la file
```

Quai_de_chargement
Réchauffement du simulateur

```
void main()
{
        float Duree_entre_deux_arrivees();
        float Duree_du_service();
        void Lire_Donnees();
        void Initialisation_Simulation();
        void Arrivee_bateau();
        void Depart_bateau();
        void Rechauffement();
        void Rapport();
        Lire_Donnees();
        Initialisation_Simulation();
```

Quai_de_chargement
Réchauffement du simulateur

```
do
{
        if(Instant_Prochain_Evenement[Arrivee] < Instant_Prochain_Evenement[Depart])
                Evenement_courant = Arrivee;          else Evenement_courant = Depart;
        Temps = Instant_Prochain_Evenement[Evenement_courant];
        if(Temps >= Duree_de_la_simulation)
        {
                Temps = Duree_de_la_simulation;
                Evenement_courant = Fin_de_simulation;
        };

        if(Temps > Instant_Prochain_Evenement[Fin_du_rechauffement])
        {
                Temps = Duree_du_rechauffement;
                Evenement_courant = Fin_du_rechauffement;
        };
```

Quai_de_chargement
Réchauffement du simulateur

```
                switch (Evenement_courant)
                {
                        case Arrivee          : Arrivee_bateau(); break;
                        case Depart           :  Depart_bateau(); break;
                        case Fin_du_rechauffement : Rechauffement(); break;
                        case Fin_de_simulation        : Rapport(); break;
                };
        } while (Evenement_courant != Fin_de_simulation);
}
float Duree_entre_deux_arrivees()
{
        // Génère la durée entre deux arrivées successives.

        return (float) (-log(1.0 - (float)((rand() % 10000) / 10000.0))
                * temps_moyen_entre_2arrivees);
}
```

Quai_de_chargement
Réchauffement du simulateur

```
float Duree_du_service()        // Génère une durée de service.
{           return (float) (-log(1.0 - (float)((rand() % 10000) / 10000.0))
                        * duree_moyenne_chargement);
}
void Lire_Donnees()
{
            printf("  Veuillez fournir les renseignements suivants : \n\n");
            printf("  Duree du rechauffement = ");
            scanf("%f", &Duree_du_rechauffement);
            printf("  Duree de la simulation = ");
            scanf("%f", &Duree_de_la_simulation);
            printf("  Temps moyen entre deux arrivees de bateaux = ");
            scanf("%f", &temps_moyen_entre_2arrivees);
            printf("  Duree moyenne d'un chargement = ");
            scanf("%f", &duree_moyenne_chargement);
            printf("  -------------------------------\n");
}
```

Quai_de_chargement
Réchauffement du simulateur

```
void Initialisation_Simulation()
{
/*          Initialise le système à vide, initialise l'horloge et tous les compteurs à 0,
            le prochain événement est une arrivée et on prévoit son instant d'occurrence      */
            Quai_Libre = true;
            Prochain_Bateau = NULL;
            Temps = 0.0;
            souche = time(NULL);
            srand((int)souche);
            Temps_total_attente_bateaux_accostes = 0.0;
            Temps_total_attente_bateaux = 0.0;
            Nombre_de_bateaux_accostes = 0;
            Instant_Prochain_Evenement[Arrivee] = Duree_entre_deux_arrivees();
            Instant_Prochain_Evenement[Depart] = (float) Maximum_des_reels;
            Instant_Prochain_Evenement[Fin_du_rechauffement] =Duree_du_rechauffement;
}
```

Quai_de_chargement
Réchauffement du simulateur

```
void Arrivee_bateau()
{
        // idem
}

void Rechauffement()
{
        Instant_Prochain_Evenement[Fin_du_rechauffement] =
                (float) Maximum_des_reels;
        Nombre_de_bateaux_accostes = 0;
        Temps_total_attente_bateaux_accostes = 0.0;
        Temps_total_attente_bateaux = 0.0;
}
```

Quai_de_chargement
Réchauffement du simulateur

```
void Depart_bateau()            // Exécution de l'événement : départ d'un bateau.
{
        struct Description_Bateau * Bateau;
        if(Prochain_Bateau  == NULL)            // Le système se vide.
        {       Quai_Libre = true;
                Instant_Prochain_Evenement[Depart] = (float) Maximum_des_reels;
        }
        else    // Le prochain bateau accoste.
        {
                Bateau = Prochain_Bateau;               Nombre_de_bateaux_accostes += 1;
                Temps_total_attente_bateaux_accostes += Temps -
                        Bateau -> Instant_Arrivee_Bateau;
                Temps_total_attente_bateaux += Temps -
                        max(Bateau -> Instant_Arrivee_Bateau, Duree_du_rechauffement);
                Instant_Prochain_Evenement[Depart] = Temps + Bateau -> Duree_Service;
                Prochain_Bateau = Bateau -> Bateau_Suivant;             free(Bateau);
        }
}
```

Quai_de_chargement
Réchauffement du simulateur

```
void Rapport()        // Imprime la durée de la simulation et l'attente moyenne par bateau.
{
        struct Description_Bateau * Bateau;
        printf("  Nombre de bateaux ayant accostes : %d\n", Nombre_de_bateaux_accostes);
        printf("  Attente moyenne par bateau : %.3f\n",
                  Temps_total_attente_bateaux_accostes / Nombre_de_bateaux_accostes);
        Bateau = Prochain_Bateau;
        while (Bateau != NULL)
        {
                  Temps_total_attente_bateaux += Temps -
                            max(Bateau -> Instant_Arrivee_Bateau, Duree_du_rechauffement);
                  Bateau = Bateau -> Bateau_Suivant;
        };
        printf("  Longueur moyenne de la file d'attente : %.3f\n",
                  Temps_total_attente_bateaux /
                  (Duree_de_la_simulation - Duree_du_rechauffement));
}
```

Quai_de_chargement
Réchauffement du simulateur

Période de simulation : [0, 50000]

	ESSAI 1	ESSAI 2	ESSAI 3
Temps moyen inter-arrivées :	10	10	10
Durée moyenne d'un chargement :	9	9	9
Nombre de bateaux ayant accosté :	5313	5327	5405
Longueur moy. de la file d'attente:	6.706	6.159	8.985
Attente moy. par bateau :	63.110	57.806	82.341

Période de simulation : [10000, 50000]

	ESSAI 1	ESSAI 2	ESSAI 3
Temps moyen inter-arrivées :	10	10	10
Durée moyenne d'un chargement :	9	9	9
Nombre de bateaux ayant accosté :	4292	4390	4251
Longueur moy. de la file d'attente:	8.331	9.098	7.659
Attente moy. par bateau :	77.776	85.046	71.792

Quai_de_chargement
Modèle M/M/1 (formules analytiques)

Soit
- ρ = taux d'utilisation = $\frac{\text{durée moyenne de service}}{\text{temps moyen entre 2 arrivées}}$ = 9/10= 0.9

- longueur moyenne de la file d'attente = $\rho^2/(1-\rho)$ = 8.1

- attente moyenne = temps moyen entre 2 arrivées * longueur moyenne de la file d'attente = 81.
- Temps moyen passé dans le système par bateau (W) = temps moyen passé dans la file (W_q) + durée moyenne de service
- Notre simulateur fournit un estimé de W_q.
  - Notre programme n'estime pas vraiment ces valeurs théoriques, à cause du biais initial!
  - Si l'on fait n essais indépendants, et que l'on utilise les n valeurs observées pour calculer un intervalle de confiance pour W_q, il n'est pas valide!
  - Lorsque $n \rightarrow \infty$, la moyenne des n valeurs converge, mais pas vers W_q.

Avantages et inconvénients d'un modèle analytique

- définition concise du problème

- solutions exactes

- facilité avec laquelle on peut évaluer l'impact d'un changement des données d'entrée sur les résultats

- la possibilité (dans certains cas) de calculer une solution optimale

- hypothèses irréelles

- formulations mathématiques complexes

Avantages et inconvénients d'un modèle de simulation

- Possibilité de décrire des systèmes complexes
- Possibilité de faire des expériences avec :
  des systèmes qui n'existent pas encore ou
  des systèmes existants sans altérer leur fonctionnement

- Simulateur comme outil d'entraînement

- Ne génère pas de solution précise

- Un changement dans les données d'entrée ⇒ une autre simulation
- Les simulateurs de systèmes complexes peuvent être coûteux à concevoir et à utiliser

- Cela peut être difficile de vérifier la validité du modèle:
  sa correspondance avec le système

Quand utiliser la simulation?

- lorsque les hypothèses exigées par un modèle analytique ne sont pas suffisamment satisfaites en réalité.
- lorsque le modèle analytique ne peut être résolu.
- BEAUCOUP D'APPLICATIONS SATISFONT À CES CONDITIONS.

Organisation générale d'un simulateur pour un modèle à événements discrets avec vision par événements

inclusion de plusieurs librairies
déclaration des données d'entrée
float Temps; // Valeur courante du temps simulé
enum Evenement { types d'événements }; // Ensemble des types d'événements
enum Evenement Evenement_courant; // Prochain événement à traiter
float Instant_Prochain_Evenement[nombre de types d'événements];
// Instant d'occurrence du prochain événement prévu de chaque type
float Duree_de_la_simulation; // Durée de la simulation
float Duree_du_rechauffement; // Durée du réchauffement
déclaration des variables d'état
déclaration des variables statistiques

Organisation générale d'un simulateur pour un modèle à événements discrets avec vision par événements

```
void main()
{
        déclaration des en-têtes des fonctions autres que celle-ci;
        Lire_Donnees();
        Initialisation_Simulation();
        do
        {
                déterminer le prochain événement (Evenement_courant) à partir de
                                                        Instant_Prochain_Evenement;
                mettre à jour l'horloge (Temps);
                appeler la fonction permettant de gérer l'événement courant;
        } while (Evenement_courant != Fin_de_simulation);
        Rapport();
}
```

Organisation générale d'un simulateur pour un modèle à événements discrets avec vision par événements

Déclarations des fonctions de générations de valeurs aléatoires.

```
void Lire_Donnees()
{
        Lecture de la durée de la simulation, de la durée de réchauffement,
        des paramètres des lois de probabilité régissant le comportement aléatoire du
        système, des paramètres du modèle décrivant le système.
}
void Initialisation_Simulation()
{
        Temps = 0.0;
        initialiser l'état du système et les variables statistiques
        initialiser les générateurs de valeurs aléatoires
        prévoir les prochains événements à survenir
}
```

Organisation générale d'un simulateur pour un modèle à événements discrets avec vision par événements

Pour chaque type d'événement propre au système simulé, déclarer une procédure ayant la forme générale suivante:

```
void Nom d'un événement()
{
        Mettre à jour l'état du système
        Mettre à jour les variables statistiques
        Générer les événements futurs
}
```

Pour l'événement Fin_de_rechauffement, déclarer la procédure suivante:

```
void Rechauffement()
{
        Remettre les variables statistiques à 0.
        Faire en sorte que cet événement ne soit plus cédulé.
}
```

Organisation générale d'un simulateur pour un modèle à événements discrets avec vision par événements

Pour l'événement Fin_de_simulation, déclarer la procédure suivante:

```
void Rapport()
{
        Finaliser le calcul des variables statistiques.
        Imprimer sous forme de rapport les résultats statistiques.
}
```