TP N°2 – IMPLEMENTATION DE LA METHODE « PERT »

Module : Conduite de Projet

REALISE PAR

Hamdi Skander
Groupe : 03

Date de remise : 07 Janvier 2018

Objectif du TP :

Le but de ce 2eme TP est d'implémenter et réaliser la méthode PERT (Program Evaluation and Review Technique).

PERT est une méthode conventionnelle utilisable en gestion de projet, planification développée aux États-Unis par la marine américaine dans les années 1950.

Elle fournit une méthodologie et des moyens pratiques pour décrire, représenter, analyser et suivre de manière logique les tâches et le réseau des tâches à réaliser dans le cadre d'une action à entreprendre ou à suivre.

La méthode PERT sert à définir le chemin critique, qui contient une série des tâches critique qui n'accepte pas de retard.

Donc, j'occupe la réalisation d'une interface graphique qui contient un espace de dessin et un ensemble de boutons qui réalisent le travail.

Ce travail sera divisé en deux partie :

- Partie I : Des explications sur le code source et les méthodes utilisées.
- Partie II : Des captures d'écrans des différentes étapes pour exécuter la méthode PERT.

Je commence par la définition des classes que j'ai utilisées :

Classe	Les attributs ou les propriétés
public class step	public int stepNumber; public int earliestStart; public int latestStart; public int margin; public double coordX; public double coordY;
public class Task	public int taskDuration; public String taskName; public int taskLevel; public List<Task> antecedent; public List<Task> successors; public step startStep; public step finishStep ; public bool painted ;
public class pert_implementation	public List<Task> projectTasks; public step lastStep;
class MainClass	**Contient seulement la méthode Main**

public partial class MainWindow	**La classe de liaison entre le métier et IHM**
	public const int width = 50 ;
	public const int height = 50;
	public static Color black = new Color(0, 0, 0);
	public static pert_implementation pert;
	public static List<Task> criticalPath = new List<Task>();
	public static List<Task> loadedTasks = new List<Task>();

Plus de détails dans **les commentaires.**

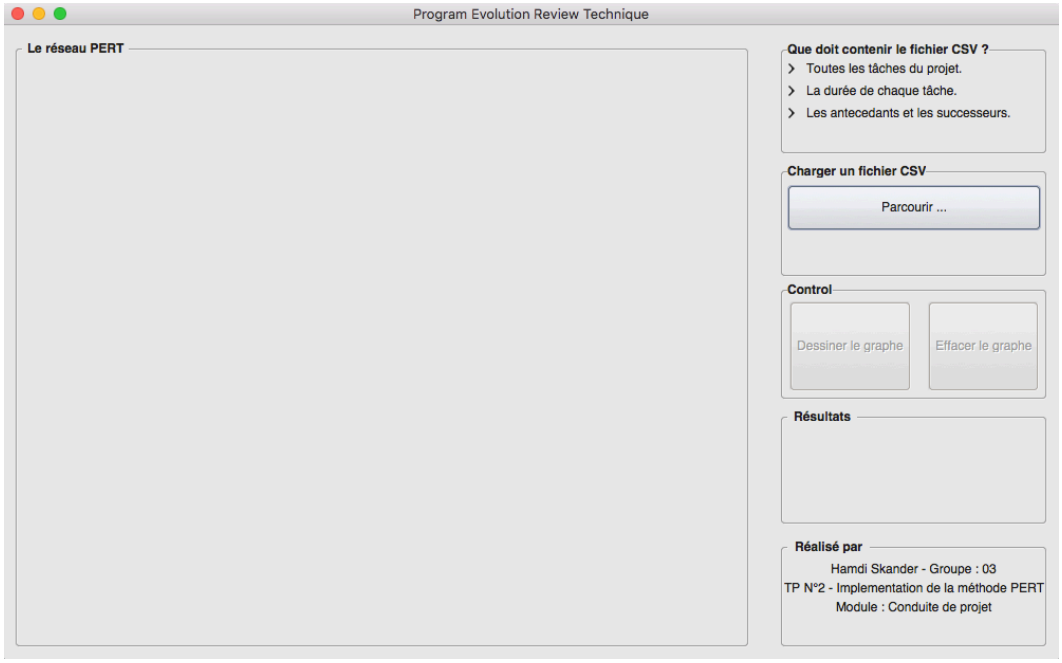
Maintenant, j'explique les méthodes que j'ai utilisées :

Classe	Les méthodes
public class step	public step(int number) : **1er constructeur** public step(int number, int delay) : **2eme constructeur**
public class Task	public Task(int duration , String name) : **constructeur de la classe Task** public void setAntecedants(List<Task> antecedants) : **Pour remplir la liste des antécédents d'une tache.** public void setSuccessors(List<Task> successors) : **Pour remplir la liste des successeurs d'une tache.** public void setSteps(step start, step finish) : **Pour attacher 2 étapes pour une tâche.**
public class pert_implementation	public void findLevels() : **Pour calculer les rangs des tâches.** public bool ifIssetAllAntecedantsLevel(List<Task> antecedants) : **Retourne True si les rangs des antécédents d'une tâche sont calculés.** public bool ifIssetAllAntecedantsEarliestStart(List<Task> antecedants) : **Retourne True si les dd+tot des antécédents d'une tâche sont calculés.** public bool ifIssetAllSuccessorsLatestStart(List<Task> successors) : **Retourne True si les dd+tard des successeurs d'une tâche sont calculés.** public int findMaximum(List<int> datesArray) : **Retourne un entier qui est le maximum dans une liste d'entiers.** public int findMinimum(List<int> datesArray) : **Retourne un entier qui est le minimum dans une liste d'entiers.** public int getLevelsCount() : **Retourne le rang maximum.** public void findCoordByLevel(int maxWidth) : **Générer des coordonnées x,y pour les taches en basent sur les rangs.** public void setEarliestStart() : **Pour calculer les dates de début au plus tôt.** public void setLatestStart() : **Pour calculer les dates de début au plus tard.** public bool isPaintedAllAntecedents(List<Task> antecedent) : **Retourne True si tous les antecedants d'une tâche sont dessinés.** public void switchSteps() **. Le début des successeurs d'une tache est le même que la fin de cette tâche.** public void switchStepsAnteced() : **La fin de l'antecedent est la même que le début de la tâche courante.** public void findMargins() : **Pour calculer la marge totale de chaque tâche.** public List<Task> findCriticalPath() : **Pour trouver le chemin critique.** public PointD MidPoint(PointD pt1, PointD pt2) : **Pour trouver le point centre situé dans la distance(pt1,pt2).**
class MainClass	**La méthode Main seulement.**

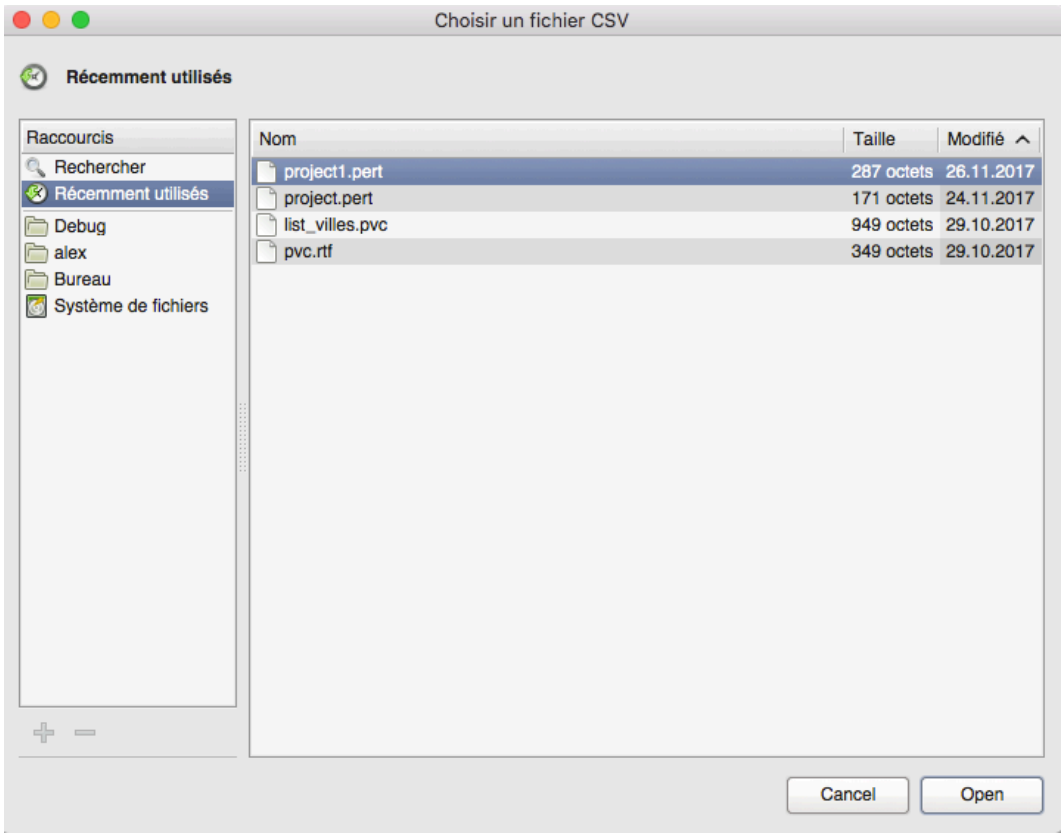
public partial class MainWindow	public MainWindow() : base(Gtk.WindowType.Toplevel) : **Constructeur de la classe partielle.** void OnExpose(object sender, ExposeEventArgs args) : **La méthode responsable de dessin graphique de graphe (etapes+taches).** public void start(object sender, EventArgs e) : **Méthode attachée au évènement de clique sur le bouton Dessiner le graphe et qui va afficher le chemin critique et la durée total du projet et attache l'événement Expose au espace du dessin.** protected void OpenControlFile(object sender, EventArgs e) : **La méthode attachée au évènement de clique sur le bouton Parcourir … pour choisir un fichier CSV et faire toutes les initialisations nécessaires et calcule toutes les dates et le chemin critique.** protected void eraseGraphe(object sender, EventArgs e) : **Pour effacer le graphe actuel et initialiser toutes les variables pour faire le chargement d'un autre fichier.**

Plus de détails dans **les commentaires.**

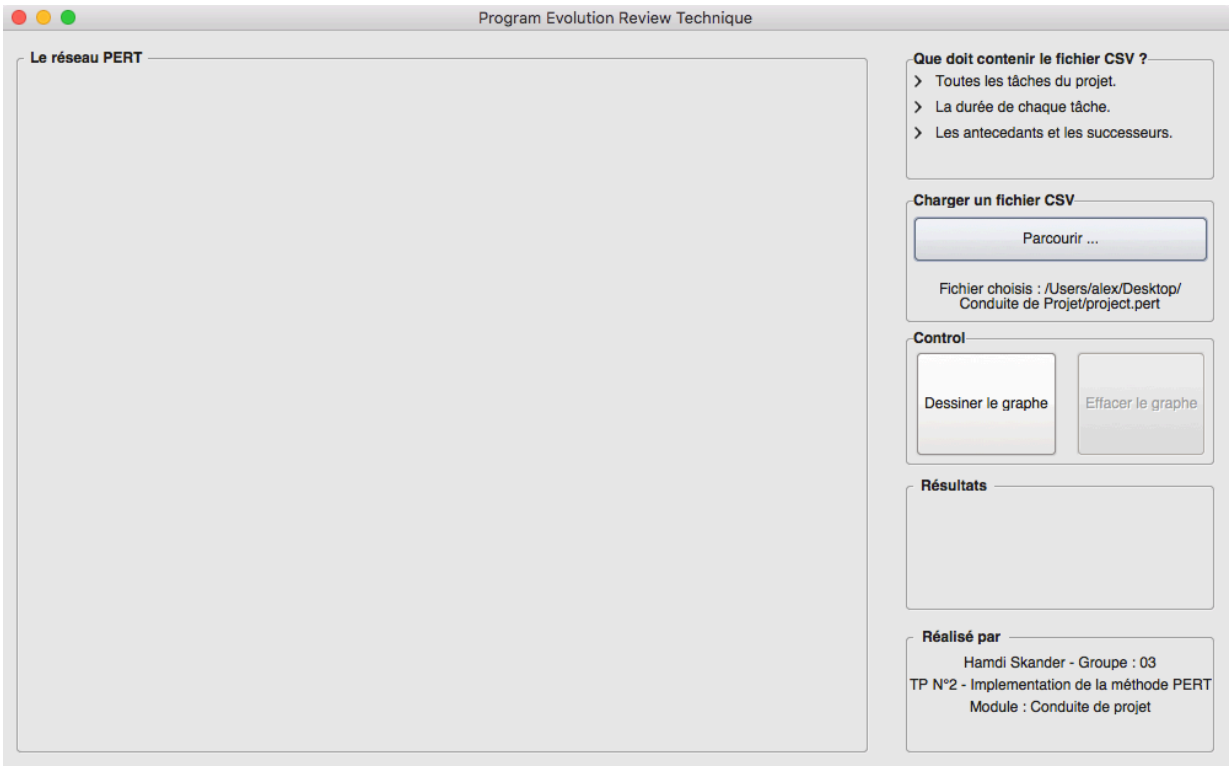
Remarque : Il existe **deux fichiers CSV** pour le teste **project.pert** et **project1.pert**



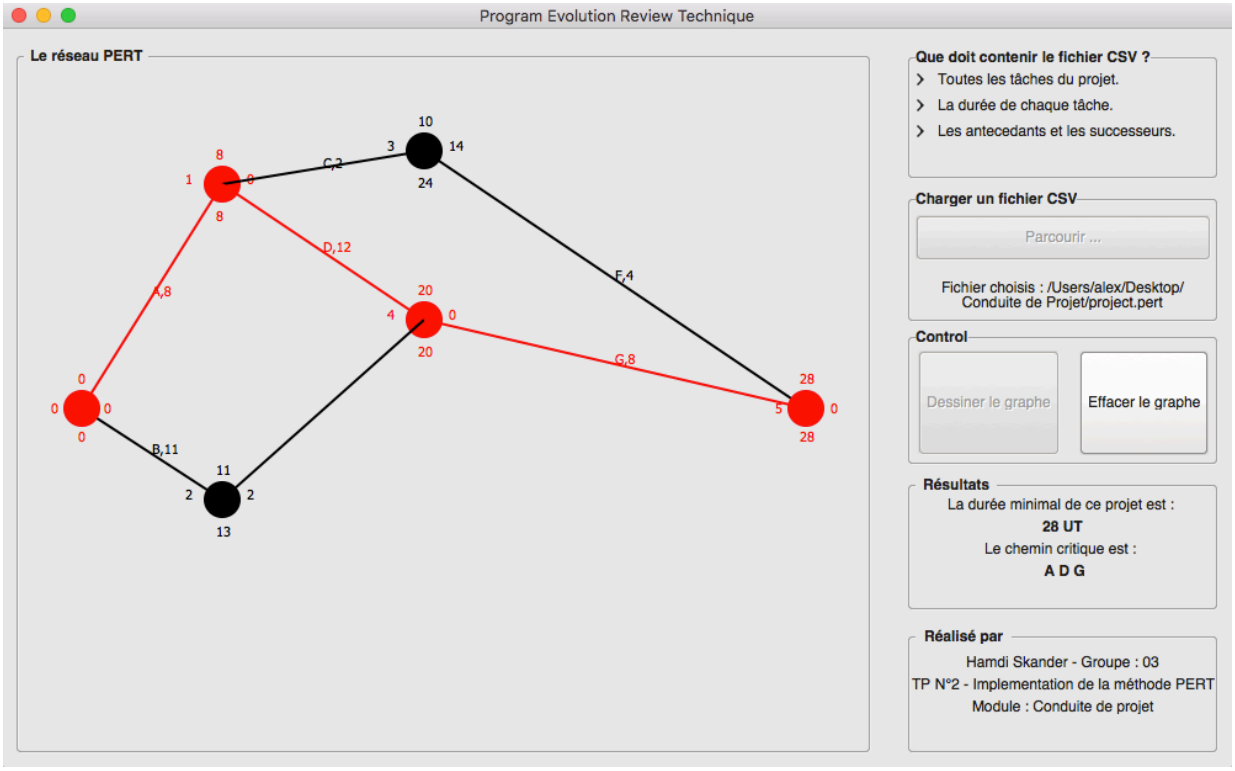
La première exécution



On clique sur Parcourir … pour choisir un fichier csv.



Maintenant on clique sur le bouton Dessiner le graphe pour afficher les résultats.



Voici le réseau PERT et le chemin critique en rouge et la durée total minimal du projet.