Projet licence informatique

TER

HLIN405

Ascenseurs fous

PEREZ Dorian
LAVAL Raphaël
DJEBLI Rayan
MAURIN Arnaud

Introduction

Dans le cadre du projet en deuxième année de licence, nous avions à implémenter un outil de simulation d'un ensemble d'ascenseurs situés dans un même bâtiment.
Nous pouvions utiliser n'importe quel langage de programmation.

Nous avons donc élaboré une application en C++ qui permet de modifier facilement l'agencement des ascenseurs grâce à un fichier de configuration, et qui retourne à la fin de la simulation le temps d'attente moyen sur toute la population, le temps d'attente minimum et maximum ainsi que le temps d'attente total.
La librairie graphique utilisée pour effectuer ce projet est SDL.

Pour ce faire, nous avons programmé des réunions avec le professeur référent, M. Montassier, tous les Mercredi à 14h.
Pour la communication entre membres du groupe, nous avons décidé d'utiliser Skype et pour la partie développement, nous avons utilisé un Git.

Sommaire

1. Description du projet

De nos jours, il existe énormément de grands buildings contenant plusieurs bureaux, hôtels, restaurants, centres commerciaux, etc… Ces lieux peuvent avoir une affluence de plusieurs milliers de personnes par jour, que ça soit pour travailler toute la journée, pour un passage d'une heure, pour une simple promenade dans les centres commerciaux ou pour un dîner par exemple.

Prenons l'exemple de la Shanghai Tower, avec 128 étages pour 106 ascenseurs, et le dernier étage se trouve à 561,3 mètres de hauteur. Les ascenseurs sont les plus rapides du monde avec une vitesse 20,6 mètres par seconde, soit un peu plus de 74 km/h.
L'ascenseur met moins d'une minute, 53 secondes exactement, pour atteindre le 119ème étage panoramique

Il est donc nécessaire et normal de se poser les questions suivantes :

Comment calculer le temps moyen d'attente de toutes les personnes présentes dans l'immeuble lorsqu'elles prennent l'ascenseur et comment optimiser au maximum celui-ci ?

Plusieurs paramètres sont à prendre en compte comme par exemple le nombre d'étages du building, les étages à forte affluence, les horaires d'affluence, la vitesse des ascenseurs, et bien d'autres encore.
Nous avons donc décidé de se focaliser dans un premier temps sur la recherche, voir s'il existait déjà des algorithmes optimisés, des plans de building, voir comment étaient organisés leurs ascenseurs etc...

Notre projet est une application qui simule le fonctionnement des ascenseurs d'un bâtiment pendant une journée.
La population commence à arriver à 8 heures et l'immeuble est vide à 20 heures. Entre temps, la population effectue un trajet basique par défaut, elle commence par les étages les plus bas et monte de plus en plus, et une fois tout en haut, redescend. Chaque personne aura un trajet aléatoire, en respectant la consigne voulant qu'il doit en premier lieu monter puis ensuite redescendre. Aucune personne ne pourra monter à un étage, descendre à un autre puis remonter.

L'objectif de cette application est, qu'à la fin de la simulation, on puisse analyser les temps d'attente des usagers.

Pour effectuer plusieurs tests en changeant la disposition des ascenseurs, la population reste la même, (le parcours des personnes) et est sauvegardée dans le fichier de configuration (population.config).
Cependant nous pouvons générer une population aléatoire (parcours) en supprimant ce fichier de configuration.
L'utilisateur peut alors effectuer des tests comme il veut avec la même population ou en générer une nouvelle pour analyser le changement du temps d'attente en fonction des ascenseurs.

2. Organisation du projet

2.1 Organisation du travail

L'organisation du travail était la partie la plus difficile dans ce projet, en effet il y avait la séance de rattrapage du premier semestre pendant les horaires de projet, ce qui a pertubé un bon nombre de réunions. Ce problème a été un grand handicap sur la répartition du travail.
Par ailleurs, les réunions avec le professeur référent étaient programmées le mercredi à 14 heures.

Ce qui nous a pris le plus de temps étaient les problèmes de compatibilité avec le système d'exploitation macOS de la librairie SDL.
Finalement, nous avons décidé de tous travailler sous Windows.

Finalement, la partie codage et débogage nous a aussi demandé du temps à cause de la simulation. Il nous a fallu beaucoup de concentration et de patience.
Nous nous sommes finalement répartis les tâches comme ceci :

- Dorian : Analyses/test du programme
- Rayan : Développement du programme
- Arnaud : Développement du second algorithme
- Raphaël : Analyses/test du programme, organisation du projet

2.2 Choix des outils de développement

Après quelques recherches, nous avons, au départ, prévu de programmer en python.
Ceci dit, tous ne connaissaient pas forcément ce langage au sein du groupe.

Rayan a donc programmé une première version en C++, afin de pouvoir travailler sur les algorithmes. Version qui aurait dû ensuite être traduite en Python.
Nous nous sommes vite rendu compte que cette première version était suffisamment avancée et que la traduire en Python serait long et compliqué. Nous sommes donc restés sur le C++.

Pour développer une simulation, il était nécessaire d'avoir une interface graphique simple à mettre en place mais efficace. Nous avons donc choisi la librairie SDL (Simple DirectMedia Layer).
Son API est utilisée pour créer des applications multimédias en deux dimensions pouvant comprendre du son, comme les jeux vidéo, les démos graphiques, les émulateurs, etc. Sa portabilité sur la plupart des plateformes et sa licence zlib, très permissive, contribuent à son succès.

La SDL gère :

- l'affichage vidéo ;
- les événements ;
- l'audio numérique (en concurrence sur ce point avec OpenAL) ;
- la gestion des périphériques communs comme le clavier et la souris mais aussi le joystick;
- la lecture de CD-Audio ;
- le multithreading ;
- l'utilisation de timers précis.

Cette librairie nous a vraiment facilité la tâche et a été très efficace.

3. Fonctionnalités

Dans cette simulation, les personnes sont représentées par des points noirs s'ils sont en mouvement, et par des points rouge s'ils sont en repos (visite d'un étage).
Les ascenseurs sont représentés en gris, une colonne correspond à un ascenseur, et une ligne correspond à un étage.

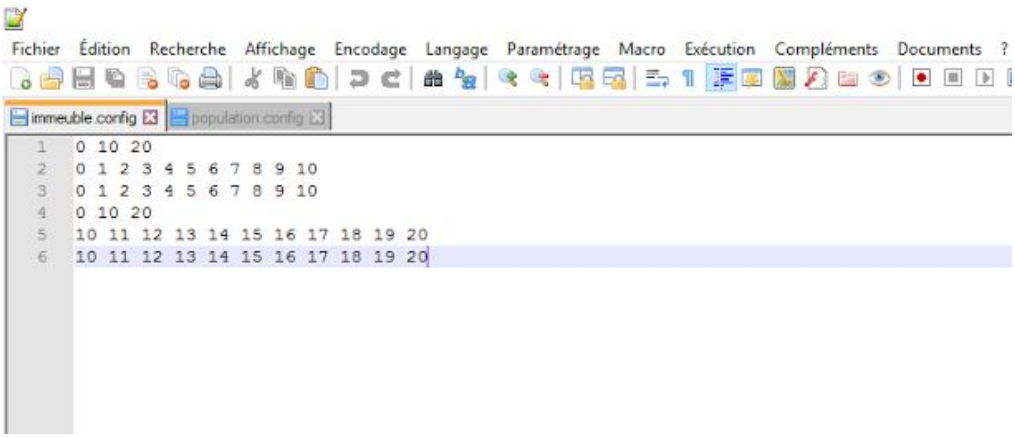
Lorsqu'un ascenseur est représenté à un étage, c'est qu'il peut y aller.
Lorsqu'il est ouvert, il est blanc et en mouvement, on peut le suivre avec un rectangle rouge. (annexe)

Les étages sont numérotés sur la gauche et le nombre de personnes se trouvant dans l'étage sont à droite.
En bas à gauche, se trouve l'heure et la vitesse de la simulation.
On peut la modifier grâce aux boutons + et - et mettre la simulation en pause grâce au bouton pause en bas à gauche.
En passant la souris sur une personne, on peut voir son identifiant, son chemin (les étages où elle va se rendre), et si elle est en mouvement, au repos ou en attente.

En passant la souris sur un ascenseur, une fenêtre s'ouvre et on peut y trouver :

- le nombre de personnes dans l'ascenseur
- le nombre de personnes se dirigeant vers l'ascenseur
- vers quel étage l'ascenseur se dirige
- la colonne où se trouve l'ascenseur
- vers quels étages l'ascenseur peut aller
- la liste des personnes dans l'ascenseur

Pour configurer l'emplacement des ascenseurs, il suffit de faire comme ceci :



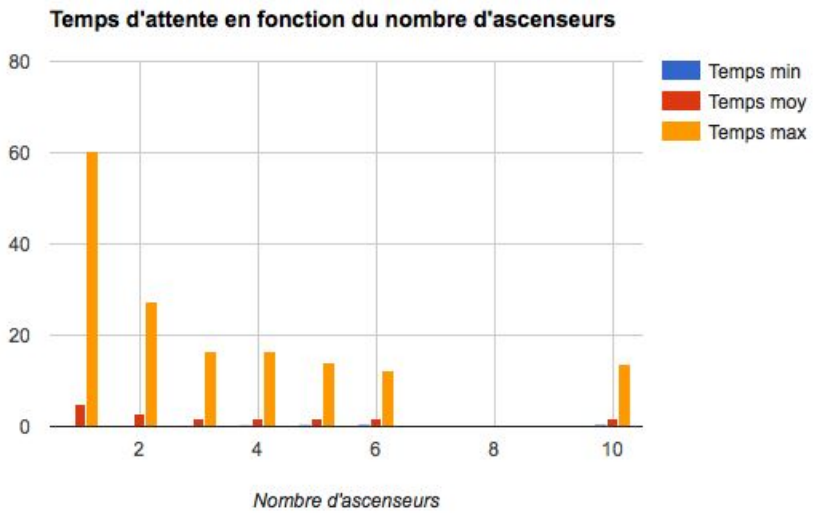
Ici, le premier ascenseur pourra desservir les étages 0, 10 et 20.
Le deuxième desservira les étages 0 à 10 et ainsi de suite.

Au début de la simulation, les personnes n'arrivent pas toutes en même temps mais par tranches horaires d'affluence.
L'arrivée des personnes s'effectuent entre 8 heures et 9 heures 30 et elles commencent à repartir vers 18 heures.
Nous sommes restés dans l'optique où les personnes passaient leur journée dans le bâtiment.

4. Analyse

Après plusieurs simulations, nous sommes arrivés à plusieurs résultats.
En effet, en changeant l'agencement des ascenseurs et le nombre d'ascenseurs, le temps d'attente moyen des usagers varie énormément.
Pour effectuer ces simulations, nous nous sommes basés sur une population de 1333 personnes, soit la population moyenne d'un immeuble de 20 étages.
La première simulation avait pour but de déterminer si augmenter le nombre d'ascenseurs fait varier le temps d'attente proportionnellement ou non.

Effet de l'augmentation du nombre d'ascenseurs				
Ascenseurs	Temps min (s)	Temps moy(s)	Temps max(s)	Temps total (s)
1	0,06	4,81	60,18	10169,2
2	0,12	2,68	27,42	5685,5
3	0,24	1,92	16,5	4219,03
4	0,3	1,77	16,5	3873,07
5	0,36	1,73	14,16	3823,2
6	0,42	1,71	12,3	3791,62
10	***0,42***	***1,79***	***13,62***	***3849,88***



Dans cette simulation, les ascenseurs desservent tous les étages.

On remarque qu'en passant de 1 à 2 ascenseurs, on réduit de moitié le temps d'attente moyen des usagers.
Néanmoins, à partir de 4 ascenseurs, le temps d'attente reste presque le même, on peut donc conclure que 4 ascenseurs suffisent pour optimiser le temps d'attente avec cette configuration.

Une autre simulation servait à voir quel est le temps d'attente moyen avec comme configuration d'ascenseur : 1 ascenseur qui dessert tous les étages, 4 autres ascenseurs qui desservent les étages 5 par 5 (de 0 à 5, 5 à 10, 10 à 15 et 15 à 20). Cette disposition, appelée Sky Lobby, est très utilisée dans les gratte-ciels des Etats-Unis.

Ascenseurs	Temps min	Temps moy	Temps max	Temps total
0 à 20 (5 par 5)	0,36	3,48	85,44	7520,63

Cette configuration reste raisonnable et reste valable pour une grande population.
C'est aussi économiquement raisonnable car on utilise moins de cages d'ascenseurs en théorie que la simulation précédente.

Nous voulions aussi savoir si la limite du nombre de personnes dans un ascenseur fait varier le temps d'attente.

Effet du "NOMBRE_PERSONNE_MAX" d'un ascenseur				
Ascenseurs	Temps min	Temps moy	Temps max	Temps total
1 (10 personnes)	0,06	4,91	59,03	10173,2
1 (20 personnes)	0,06	4,81	60,18	10169,2

Ici, nous remarquons clairement que le nombre maximum de personnes ne semble pas avoir d'impact. L'attente perdue par le nombre d'allers est compensée par la fréquence plus élevée de ces derniers.
Bien évidemment, nous pouvons faire un nombre incalculable de tests pour définir encore plus précisément la configuration optimale pour un bâtiment voulu.

5. Développement

Nous voulions avoir une première démo rapidement pour commencer à mieux se projeter dans la suite du projet. Pour cela nous avons décidé dans un premier temps de développer en Python. Cependant, certains d'entre nous ne connaissaient pas encore le langage.
Nous avons donc opté pour, dans un premier temps, développer en C++ via la librairie SDL (en raison des connaissances que nous possédions déjà dessus) puis de traduire la démo en Python. Mais la démo ayant eu une grande avancée la semaine suivante, nous avons finalement décidé de rester sur ce programme.
Pour résumer, nous avons conservé le C++, car nous avions tous des connaissances en ce domaine, et SDL car la personne ayant développé la démo a utilisé cette librairie en raison de lacunes sur SFML et pour gagner du temps.

En ce qui concerne les algorithmes, c'est assez simple.

On parcourt notre liste de personnes. Si une personne attend un ascenseur, on "lance" notre algorithme.
Premièrement, on regarde s'il existe un parcours entre la personne et sa destination. Dans le cas échéant, chaque parcours est stocké.
On parcourt alors notre liste de parcours. Et on vérifie que le premier ascenseur n'est pas complet. Si c'est le cas, on vérifie qu'il est disponible à notre étage. Si tout est bon, on l'associe à la personne.
Sinon on se met à vérifier que l'ascenseur passe devant l'étage de la personne ou qu'il est vide et qu'il n'attend personne.
Si c'est le cas, on calcule le nombre d'étapes maximum. On compte le nombre d'étages qui sépare l'ascenseur de la personne et la personne de sa destination, tout en prenant en compte uniquement les étages desservis par l'ascenseur. Puis si ce nombre est plus petit que le précédent on garde en mémoire ce parcours.

A la fin, si nous n'avons toujours pas trouvé de parcours avec un ascenseur disponible, on utilise le plus optimisé qu'on puisse trouver (via le calcul précédent).
Si on ne trouve rien, on attend à la prochaine boucle qu'un parcours se débloque.

Après qu'on ait fait ceci pour les x personnes attendant, on parcourt la liste des ascenseurs à qui on a ordonné de se déplacer.
On fait en sorte que l'ascenseur n'ignore pas une personne pendant son parcours.
Comment trouve-t-on tous les parcours disponibles ?

De manière récursive, on trouve les ascenseurs capables d'aller au point A et/ou au point B. Si on trouve un ascenseur capable de faire les deux, l'algorithme a trouvé un parcours.
Si l'ascenseur ne peut qu'aller au point B, on le stocke et on reparcourt l'algorithme en allant du point A au point C (C étant un étage que dessert l'ascenseur).
On fait ceci de manière récursive et on stocke le tout dans un vecteur.

Ceci dit, il est possible d'optimiser cet algorithme, d'une part en ajoutant une intelligence artificielle à la population, et d'autre part en stoppant la récursivité dès qu'un trajet disponible est trouvé par exemple.

Par ailleurs, dans un cadre de test et de recherches, un second algorithme a été envisagé. Celui-ci partait d'un parti pris différent : les personnes entrant dans la salle prévoient leur itinéraire selon le trajet le plus court, sans se soucier du temps de parcours. Ainsi, les personnes attendent toujours un ascenseur bien précis, même si ce dernier mettra plus de temps qu'un autre à venir. Les ascenseurs en eux-mêmes fonctionnent sur une liste d'attente reposant sur une pile. A chaque appel à un étage/programmation d'une destination, l'ascenseur ajoute cette étape à sa pile, et la retire une fois la destination atteinte. Cet algorithme est bien plus rapide et léger que l'algorithme conservé dans la version finale, mais seulement en terme de ressources machines, car bien qu'il n'ait pas été finalisé, nous avons pu constater qu'il multipliait le temps d'attente moyen par dix, les ascenseurs et la population n'optimisant en rien leur trajets.

6. Perspectives et conclusion

6.1 Perspectives

Grâce à cette application, nous pouvons tester d'innombrables configurations pour n'importe quel type de bâtiment. Bien sûr, il faudrait améliorer encore certaines choses, car le temps d'exécution de la simulation reste encore un problème. Faire un algorithme qui permettrait de nous afficher les mêmes résultats à la fin de la simulation, mais en quelques secondes serait plus efficace. Néanmoins, le fait de voir ce qu'il se passe grâce à l'interface graphique permet de visualiser un peu plus la problématique et aussi de prévoir quels sont les erreurs à ne pas produire. Par exemple, si un ascenseur tombe en panne, s'il faut en prévoir un de secours…

La meilleure amélioration serait de pouvoir rendre le choix de la configuration de l'immeuble et de l'ascenseur plus instinctive, avec une interface graphique au

lancement de la simulation par exemple. Nous pourrions aussi développer un meilleur algorithme d'intelligence artificielle.

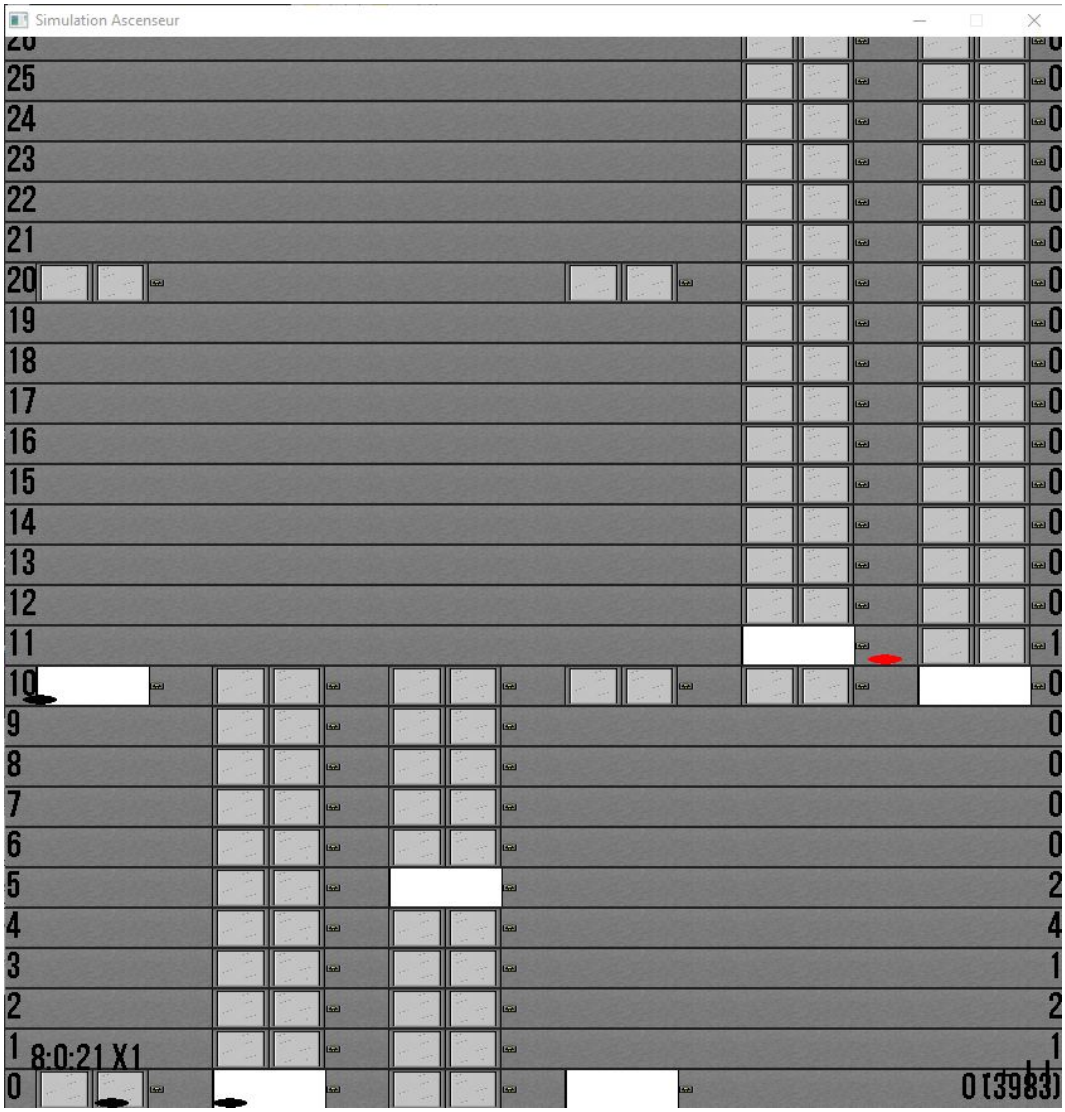
6.2 Conclusion

L'application réalisée est fonctionnelle et prête à l'emploi. Nous pouvons tester tout type de configuration et avoir son temps d'attente moyen en fonction de celle-ci.

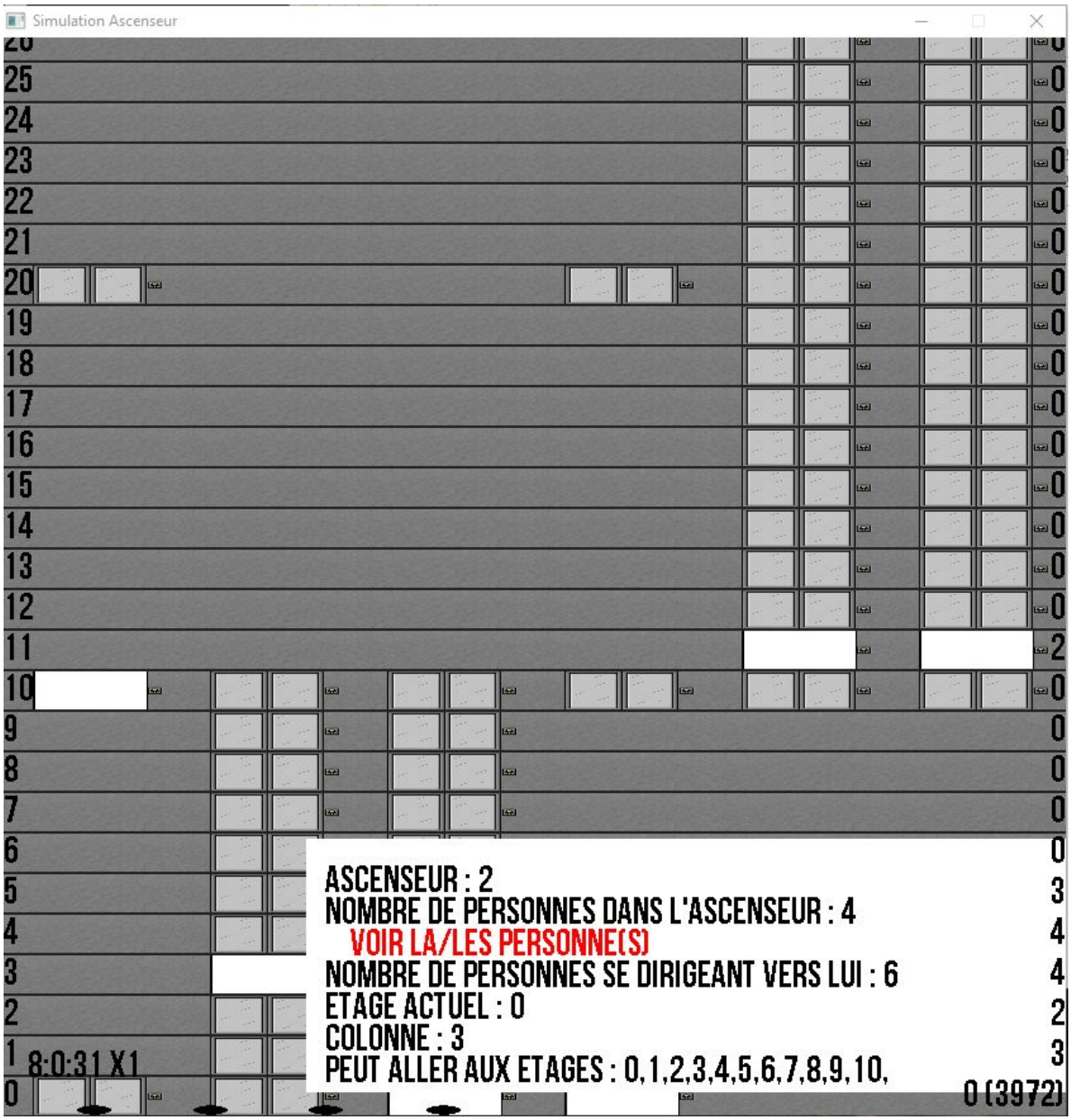
La bibliothèque SDL était la grande découverte dans notre projet, donc il nous a fallu plus de temps pour s'y familiariser et s'y adapter. On peut aussi ajouter qu'elle est très puissante en matière de graphisme. Quant à l'analyse, nous pensons qu'elle est suffisante étant donné que notre application fonctionne convenablement et obéit aux demandes du sujet.

Ce projet a été très bénéfique pour nous, c'était une expérience très enrichissante car nous avons pu travailler sur les différentes phases de la réalisation d'une application réelle de modélisation, en partant des besoins à la concrétisation des solutions répondant à ceux-ci.
Pour commencer, nous avons pu acquérir de nouvelles compétences en travaillant en collaboration entre quatre étudiants qui ne se connaissent pas, et possédaient des niveaux de compétences différents, ce qui nous a permis d'apprendre beaucoup de choses les des autres, de s'entraider, de surpasser les difficultés rencontrées tout au long de ce parcours et enfin réaliser un travail considérable en un temps remarquable.

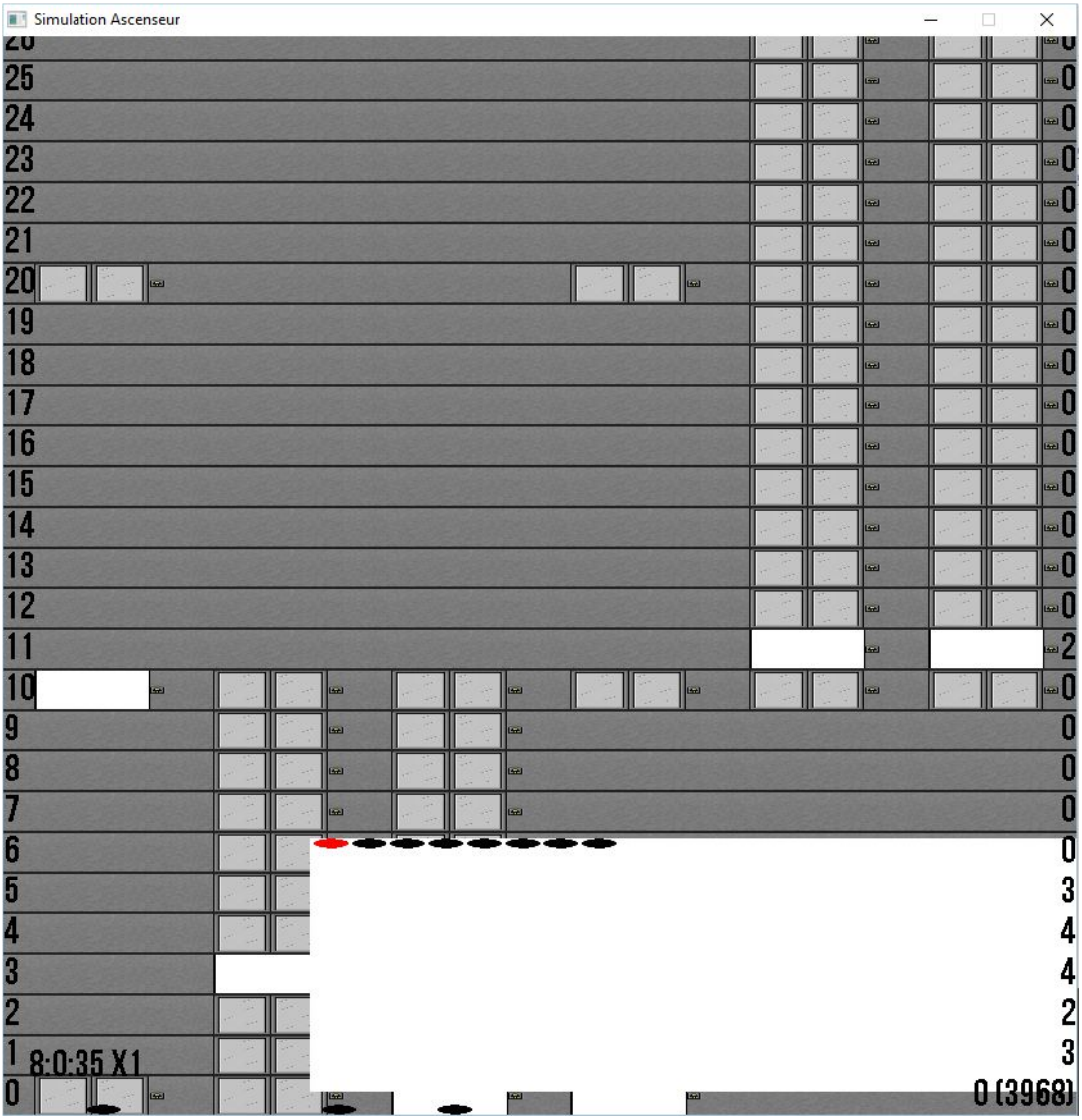
Annexe



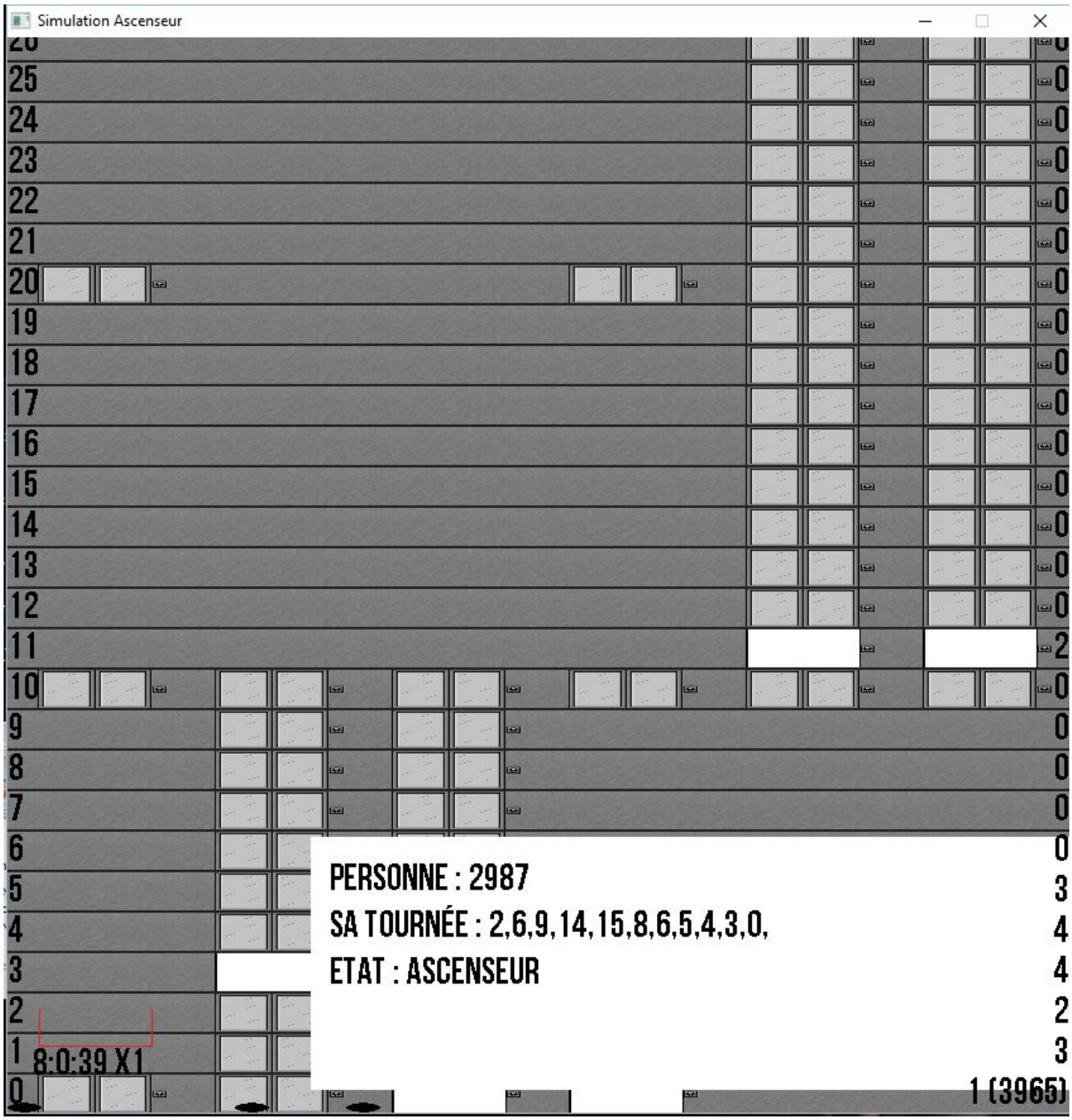
Ici, il y a 6 ascenseurs. Le premier et le quatrième desservent les étages 0, 10 et 20…



Vue en cliquant sur un ascenseur



Vue en cliquant sur voir les personnes dans l'ascenseur



Vue en cliquant sur une personne

Code source moteur.cpp (contient l'algorithme qui fait bouger les ascenseurs et les personnes)

```
void bougerAscenseurs(LoaderObject& objects, vector<Ascenseur>& ascenseurs, long tempsEcoule,
Population& population)
{
   float mouvementPx=0;
   long vitesse=0;
   int currentPos=0;
   int aimPx=0;
   int newPos;
   for (size_t i=0;i<ascenseurs.size();i++)
   {
      vitesse=ascenseurs[i].getVitesse();
      if (vitesse!=0) /// L'ascenseur est en mouvement
      {
         currentPos=ascenseurs[i].getPosition();

aimPx=objects.getSurface(ECRAN)->h-(ascenseurs[i].getEtagesAim()+1)*objects.getSurface(ASCENSEUR)->h;
         mouvementPx=(vitesse*((float)tempsEcoule)/1000)*objects.getSurface(ASCENSEUR)->h/2.5;
         if (vitesse>0)
            newPos=ceil((double)currentPos+mouvementPx);
         else
            newPos=floor((double)currentPos+mouvementPx);
         if ((vitesse>0 && currentPos+mouvementPx>=aimPx) || (vitesse<0 &&
currentPos+mouvementPx<=aimPx)) /// S'il est arrivÈ ‡ destination (ou un peu plus loin)
         {
            mouvementPx=aimPx-currentPos; /// On le positionne au bon endroit
            ascenseurs[i].setVitesse(NONE);
            ascenseurs[i].setCurrentEtages(ascenseurs[i].getEtagesAim()); /// On indique qu'il est arrivÈ ‡
destination
            gererPersonnesAscenseur(ascenseurs[i], population, objects, ascenseurs.size()); /// Et on gËre les
personnes qui sont ‡ l'intÈrieur
         }
         ascenseurs[i].setPosition(newPos);
      }
      else
      {
         gererPersonnesAscenseur(ascenseurs[i], population, objects, ascenseurs.size());
      }
   }
}


void bougerPersonnes(LoaderObject& objects, Population& population, long tempsEcoule, vector<Ascenseur>&
ascenseurs)
{
   float mouvementPx=0;
   long vitesse=0;
   int currentPosX, currentPosY;
   int aimPx=0;
   for (size_t i=0;i<NOMBRE_PERSONNES;i++)
```

```
  {
    vitesse=population.getListe()[i].getVitesse();
    if (population.getListe()[i].getVisible() && vitesse!=0) /// Si on est visible et qu'on bouge
    {
      currentPosX=population.getListe()[i].getPositionX();
      currentPosY=population.getListe()[i].getPositionY();
      aimPx=population.getListe()[i].getColsAim()*objects.getSurface(ASCENSEUR)->w;
      if (population.getListe()[i].getWaitElevators()) /// On se dirige vers la porte de l'ascenseur
      {
        aimPx+=(objects.getSurface(ASCENSEUR)->w/2)-10; /// c'est juste du graphique ! Pour donner
l'impression qu'on va bien dans l'ascenseur
      }
      mouvementPx=(vitesse*((float)tempsEcoule)/1000)*objects.getSurface(ASCENSEUR)->h/2.5;
      if ((vitesse<0 && currentPosX+mouvementPx<=aimPx) || (vitesse>0 &&
currentPosX+mouvementPx>=aimPx)) /// On est arrivÈ ‡ destination
      {
        mouvementPx=aimPx-currentPosX;
        if (vitesse>0 && population.getListe()[i].getColsAim()>=ascenseurs.size()) /// on sort de l'Ècran
        {
          population.getListe()[i].setVisible(false);
          if (population.getListe()[i].getCurrentEtage()==0) /// Il rentre chez lui
          {
            population.getListe()[i].setIsHome(true);
          }
        }
        else
        {
          if (population.getListe()[i].getWaitElevators()) /// On attends un ascenseur et on a bougÈ vers celui
ci, on est donc arrivÈ ‡ la porte de l'ascenseur voulu
          {
            population.getListe()[i].setWaitElevators(false);
            population.getListe()[i].setVisible(false); /// On est dans l'ascenseur, on n'est plus visible
            ascenseurs[population.getListe()[i].getAscenseurId()].addPersonnes(i);
            ascenseurs[population.getListe()[i].getAscenseurId()].decreaseWait();
          }
          else
          {
            population.getListe()[i].setWaitElevators(true); /// On est au "bouton" d'ascenseurs, on attends
          }
        }
        population.getListe()[i].setVitesse(0);
        population.getListe()[i].setBusy(false); /// On est plus en mouvement, on peut donc lui donner de
nouvelles directives
      }
      if (mouvementPx>0 && mouvementPx<1)
        mouvementPx=ceil(mouvementPx); /// Bon la c'est un peu de la triche, on ne peut pas positionner les
surfaces en float, uniquement en int, malheureusement, on obtiens souvent un float, donc on ceil pour arrondir
au supÈrieur
      else if (mouvementPx<0)
        mouvementPx=floor(mouvementPx); /// Idem mais en infÈrieur

      currentPosX=currentPosX+mouvementPx;
      population.getListe()[i].setPosition(currentPosX, currentPosY);
    }
```

```
    }
}
```

Bibliographie

https://www.quora.com/What-algorithm-is-used-in-modern-day-elevators
https://www.quora.com/Is-there-any-public-elevator-scheduling-algorithm-standard
Image plan :
https://upload.wikimedia.org/wikipedia/commons/e/e4/World_Trade_Center_Building_Design_with_Floor_and_Elevator_Arrangement.svg
Exemple de simulation :
http://www.commentcamarche.net/forum/affich-4866110-simulation-d-ascenceur
Autres :
http://sal.aalto.fi/publications/pdf-files/rsii97b.pdf
SDL :
https://www.libsdl.org