FROISSART Kévin p2002504
GASQUET Anton p1710949
GACHOD Valentin p1810234

Responsable :
AKNINE Samir

Université Lyon 1 - Licence 3 Informatique

SA6

-

Gestion intelligente de parkings dans un centre urbain

Rappel du projet : “L’objectif de ce projet est de proposer une méthode distribuée pour l’affectation des places de stationnement à des véhicules intelligents. Le rôle de l’étudiant est de formaliser et d’implémenter l’ensemble des comportements nécessaires aux véhicules pour interagir avec les parkings et négocier le tarif de stationnement qui est décidé dynamiquement en tenant compte d’un ensemble d’information, telles que la durée du stationnement, le profil de l’usager, le taux d’occupation du parking, le créneau horaire, etc.”

Lien du répo git : https://github.com/KevinFroissart/SA6_CarPark_AI

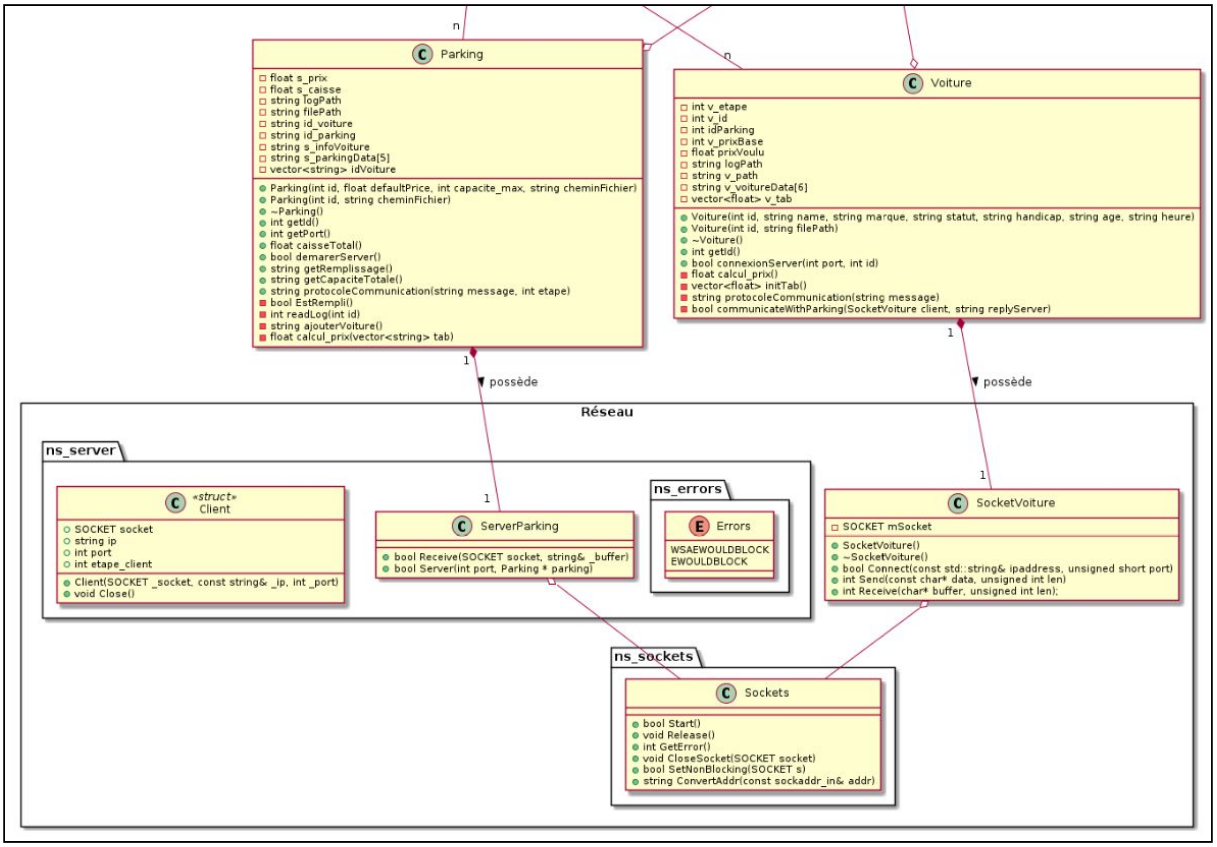
I. Conception Objet

La première décision que nous avons prise est d'avoir choisi le C++ au détriment du Java. Bien que les deux soient très bien pour la conception objet, une majorité du groupe était plus à l'aise en C++ qu'en Java. Nous avons ensuite rapidement compris comment créer nos objets et avons alors créé nos modèles Voiture et Parking. L'une des fonctionnalités que nous avons implémenté au début et qui nous a servi tout du long c'est la construction dynamique de nos objets en fonction de fichiers CSV. Voici l'exemple d'une ligne du fichier Voiture.csv :

> 9,Toyota,étudiant,non,18,2,

Nous pouvons lire ici : l'ID de la voiture, le modèle, le statut du conducteur, s'il est handicapé, son âge et enfin la durée souhaitée de stationnement. D'une manière identique, nous lisons d'autres informations propres aux Parkings dans leur CSV. Ayant alors trouvé l'usage de méthodes similaires nous avons créé le namespace ToolBox dans lequel nous retrouvons plusieurs méthodes nous permettant de lire des CSV, écrire dans ceux-ci, convertir les informations lues en tableaux, récupérer des informations spécifiques et j'en passe...

La seconde décision importante que nous avons dû prendre est liée à la manière dont nous voulions que nos Objets communiquent. Une idée, très soutenue par notre encadrant, était de les faire se parler en réseau via des sockets. Une seconde idée était de les faire communiquer via des tableaux, stockés dans des CSV. Sans surprise nous avons choisi les Sockets. C'était un peu fastidieux à mettre en place mais c'était bien plus pratique que de lire dans des tableaux. Valentin nous a fourni le code d'un ancien projet de communication Client/Server de L2. La difficulté était alors d'adapter ce code à nos besoins. Et je dois dire que cette étape nous a donné du fil à retordre. Il nous aura fallu pas moins de deux semaines avant de pouvoir faire communiquer nos Voitures et nos Parkings de manière indépendante et sans erreur.



Comme le montre ce diagramme UML (en entier dans les annexes fig 1), chaque Parking est capable de démarrer un Serveur grâce à un port individuel stocké dans Parking.csv et les Voitures sont associées à un socket qui permet alors aux Parkings et aux Voitures de communiquer entre eux grâce à la méthode `` connexionServer() `` de la classe modèle Voiture. Afin de faire se parler les Objets entre eux nous avons rencontré une autre difficulté, les threads. Après un peu de documentation et aussi grâce au cours de Programmation Concurrente, Kévin et Anton ont écrit une méthode qui permet de créer des Threads de manière dynamique en fonction du nombre de Parking et de Voiture. De la manière dont la boucle est faite, toutes les voitures cherchent de manière aléatoire un Parking et entament la discussion. Cette manière séquentielle implique que deux Voitures ne peuvent pas discuter avec un Parking au même moment, bien que ce serait possible grâce aux threads. Cela pourrait être un point à améliorer.

II. IA et Protocole

Kévin a ensuite ajouté ce que nous avons appelé un “protocole”, soit une sorte d’anti-sèche pour que les Parkings et Voitures puissent communiquer ensemble. De cette manière, quand une Voiture tente de communiquer avec un Parking, celui-ci lit et analyse le message de la Voiture et renvoie alors une réponse appropriée afin d’établir un échange sensé entre les objets. Comme le montre le logigramme (fig. 2) en annexe, la Voiture envoie ses informations au Parking, << l’ID de la voiture, le modèle, le statut du conducteur, s’il est handicapé, son âge et enfin la durée souhaitée de stationnement >> comme indiqué plus

haut ; puis le Parking calcule le prix du stationnement en fonction de ces facteurs grâce à une méthode faite par Anton. Enfin, la Voiture calcule elle aussi le prix qu'elle souhaiterait payer et si les deux montants concordent un minimum, le Parking réserve une place à la Voiture et ajoute le montant de la transaction à sa caisse. Dans le cas où le prix ne convient pas à la Voiture, celle-ci peut essayer de négocier le prix en faisant jouer à sa faveur plusieurs facteurs comme par exemple :

- La possession d'une carte handicap
- Si nous sommes en semaine
- Si nous sommes en heure creuse
- Si la Voiture est un client fidèle

Toutes ces informations sont vérifiées par le Parking et celui-ci accorde ou non un rabais à la Voiture ou alors lui propose une petite réduction. Si un Parking et une Voiture n'arrivent pas à se mettre d'accord, aucune transaction n'est effectuée et la Voiture entame une conversation avec un autre Parking. Tous ces échanges sont enregistrés et l'historique des passages est aussi pris en compte grâce à des méthodes faites par Kévin permettant d'écrire dans des CSV propres à chaque Voiture et à chaque Parking un historique de ses allées et venues dans les Parkings du centre urbain. Les conversations sont toutes stockées dans une map ayant pour clé l'ID du Parking et pour valeur une autre map ayant pour clé l'ID de la voiture et pour valeur la discussion entre ces deux entités.

III. Interface Graphique

La troisième décision importante que nous avons dû prendre a été de choisir avec quelle librairie nous voulions développer notre interface graphique. Nous avions le choix entre Qt et SFML. Notre choix s'est tout de suite porté vers Qt mais nous avons rencontré un problème majeur. En effet, impossible d'importer notre projet actuel vers un projet Qt, certains types comme les std::string devinrent des QString et nous avons ainsi rencontré beaucoup de problèmes de compatibilité. Nous nous sommes alors tournés vers SFML qui lui aussi nous a causé du tort. Certains d'entre nous, développant sur windows et utilisant le Windows Subsystem for Linux pour exécuter le code se sont retrouvés dans l'incapacité d'afficher l'interface via wsl. Nous avons réglé ce problème en utilisant XcXsrv Windows Server afin de renvoyer l'affichage X11 de Linux vers notre machine Windows. Anton et Kévin ont alors pu travailler sur l'interface graphique, sur laquelle nous pouvons retrouver comme le montre l'image (fig 3) des annexes, les différents Parking, suivi de leur taux de remplissage ainsi que les revenus amassés depuis le début de l'exécution du programme.

Comme le montre la capture d'écran (fig 3) des annexes, nous avons la possibilité de cliquer sur des boutons "Parkings" qui sont générés dynamiquement en fonction du nombre de parkings. Cliquer sur ces boutons permet d'ouvrir une nouvelle fenêtre qui affiche alors une autre liste de boutons comme le montre la capture d'écran (fig 4) des annexes. Ces boutons correspondent aux discussions stockées dans la "map" dont on parle quelques lignes plus haut. Cliquer sur l'un de ces boutons permet d'afficher la conversation entre le Parking lié à la fenêtre et les voitures avec lesquelles le parking a eu une discussion.

La gestion du multi fenêtrage avec SFML nous a posé quelques soucis. Notamment dû au fait que tout notre code de fenêtre principale se trouve dans une boucle while, qui était donc interrompue. Nous avons décidé de laisser cela ainsi. La fenêtre principale reprend là où elle en était lorsque l'on ferme la sous-fenêtre du parking.

IV. Conclusion

Si amélioré, le projet pourrait être appliqué à une gestion réelle des parkings d'une compagnie. Et ainsi, faire des économies, tout en permettant de fidéliser les usagers et faire payer le tarif le plus juste possible. Cela reste cependant fortement hypothétique. Et ce programme a plus vocation à simulation qu'autre chose.
Nous avons proposé une solution simple, qui reste au stade de simulation. Les parkings sont initialement vides, et se remplissent progressivement. Chaque voiture va trouver un parking, et interagir avec celui-ci (c'est-à-dire **négocier**), jusqu'à soit obtenir une place, soit chercher un autre parking. Les objets fonctionnent grâce à des threads et communiquent via des sockets permettant un partage rapide et efficace d'informations.

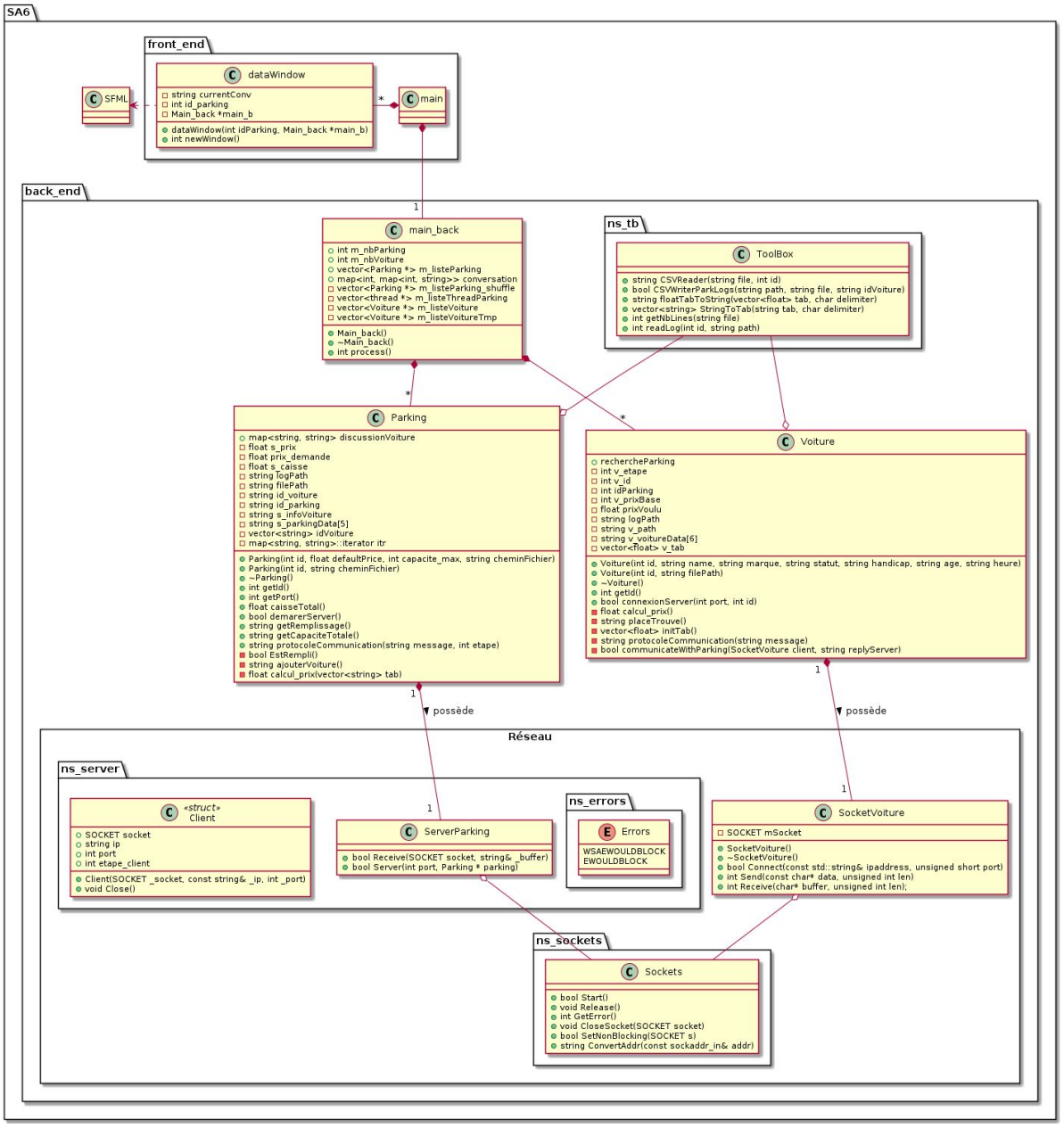
Du point de vue de l'interface, le remplissage et la caisse de chaque parking sont visibles en temps réel. Cliquer sur un parking ouvre une fenêtre affichant spécifiquement les infos de celui-ci. Notamment les détails des communications avec les voitures. Nous regrettons tout de même ne pas avoir pu faire l'interface avec Qt qui aurait été bien plus joli, ergonomique et facile à faire.

Au niveau des axes d'amélioration, nous aurions voulu ajouter une méthode permettant aux voitures de 'quitter' le parking après avoir écoulé le temps que le parking leur avait alloué. Ainsi, le parking aurait une nouvelle place de libre et la voiture pourrait se mettre à la recherche d'une autre place. Nous aurions aussi aimé ajouter un bouton "Ajouter Voiture" permettant d'ajouter automatique une ligne à notre CSV de voiture et ainsi lui permettre de rentrer à son tour dans l'algorithme et chercher une place de parking. Il serait également plus représentatif de la réalité de faire quitter les voitures du parking à l'issue du nombre d'heures voulu, et les faire revenir sur le marché après un certain délai. Voir nous aurions pu les faire rester plus longtemps que prévu, moyennant des frais supplémentaires.
Enfin, l'amélioration majeure aurait été d'avoir pu faire communiquer les voitures avec les parkings de manière **parallèle** et non **séquentielle**.

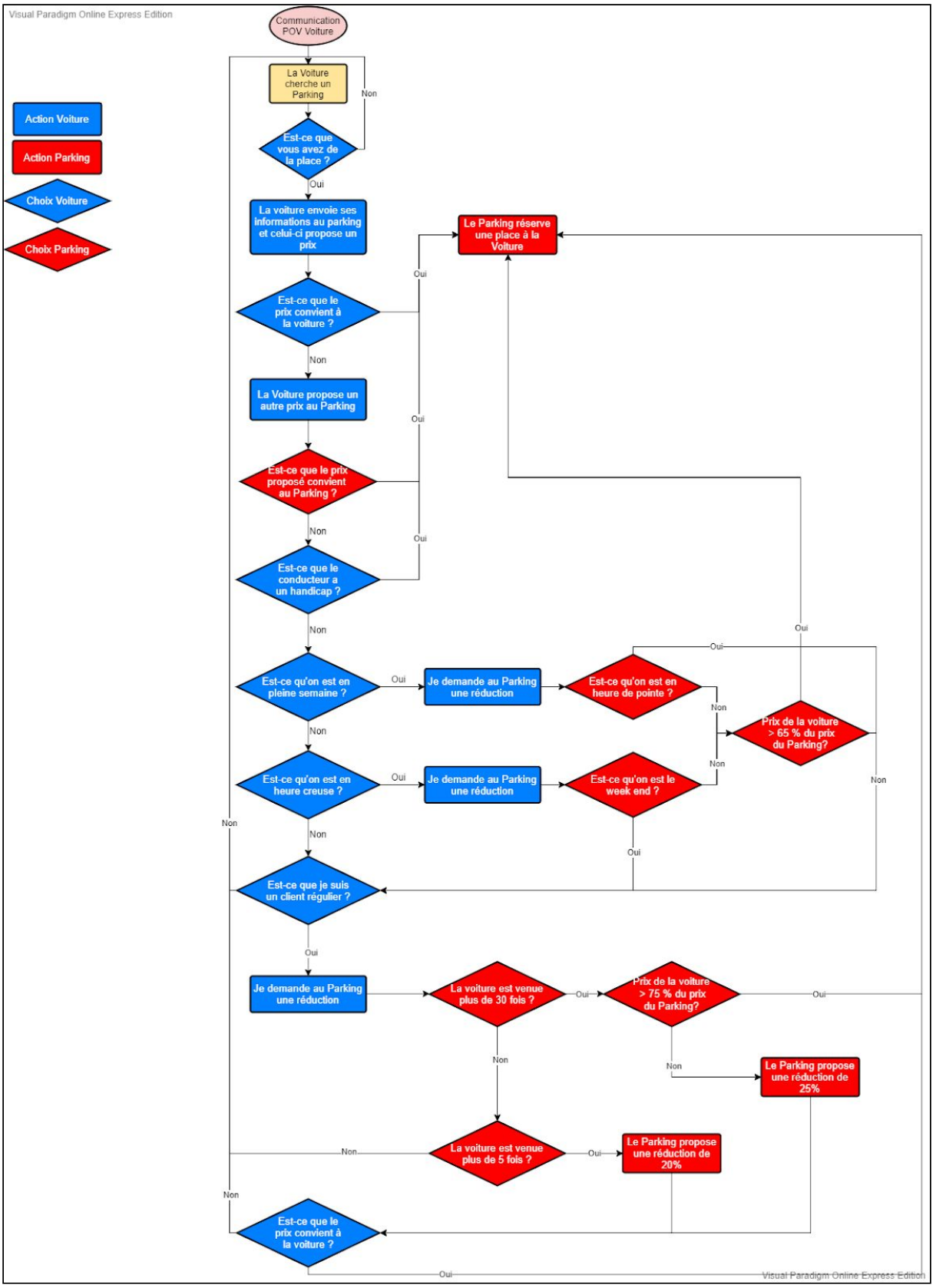
V. Annexes

Toutes les images se trouvent aussi dans le dossier SA6/doc/ du projet si vous souhaitez les consulter avec une meilleure qualité.

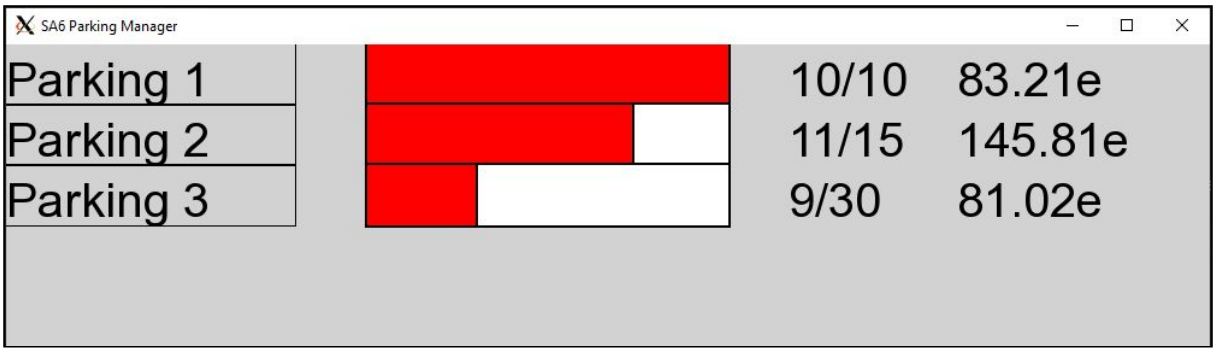
1.



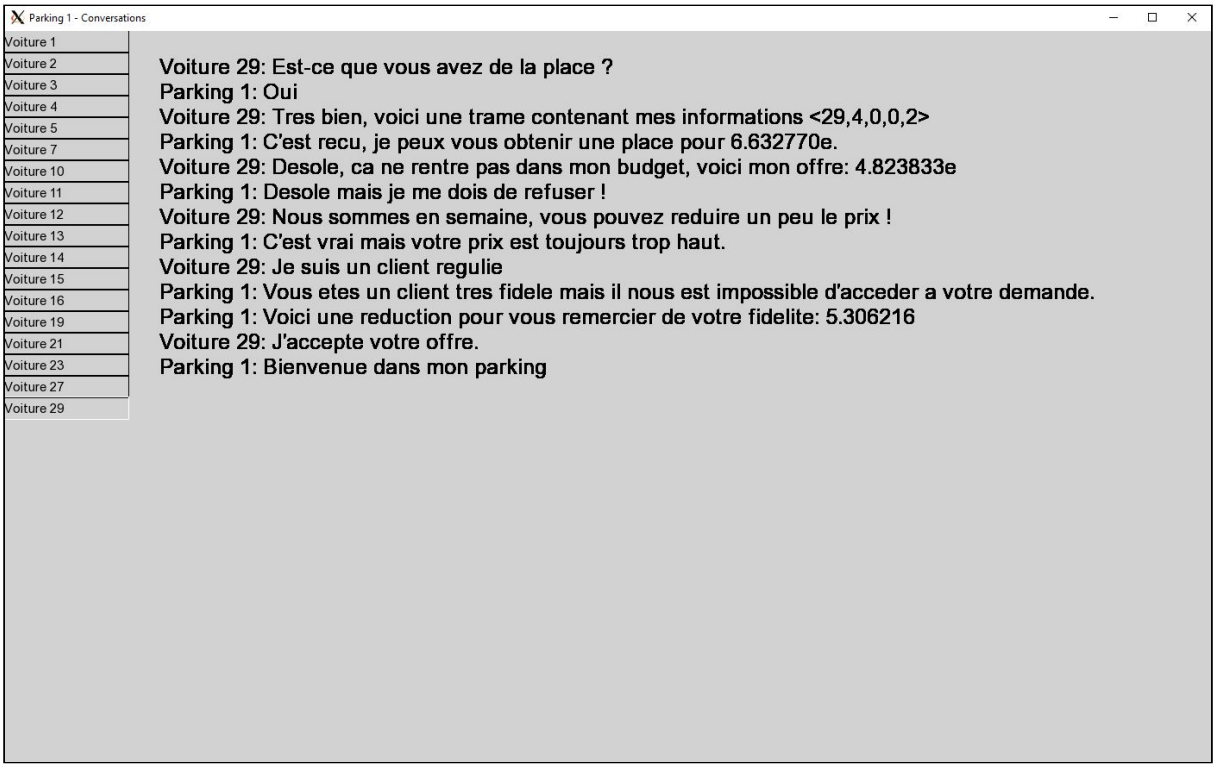
2.


Visual Paradigm Online Express Edition
Communication POV Voiture
La Voiture cherche un Parking
Non
Est-ce que vous avez de la place ?
Oui
La voiture envoie ses informations au parking et celui-ci propose un prix
Le Parking réserve une place à la Voiture
Oui
Est-ce que le prix convient à la voiture ?
Non
La Voiture propose un autre prix au Parking
Oui
Est-ce que le prix proposé convient au Parking ?
Non
Oui
Est-ce que le conducteur a un handicap ?
Non
Oui
Oui
Est-ce qu'on est en pleine semaine ?
Oui
Je demande au Parking une réduction
Est-ce qu'on est en heure de pointe ?
Non
Prix de la voiture > 65 % du prix du Parking?
Non
Non
Est-ce qu'on est en heure creuse ?
Oui
Je demande au Parking une réduction
Est-ce qu'on est le week end ?
Non
Oui
Non
Non
Est-ce que je suis un client régulier ?
Oui
Je demande au Parking une réduction
La voiture est venue plus de 30 fois ?
Oui
Prix de la voiture > 75 % du prix du Parking?
Oui
Non
Non
Le Parking propose une réduction de 25%
Non
La voiture est venue plus de 5 fois ?
Oui
Le Parking propose une réduction de 20%
Non
Non
Est-ce que le prix convient à la voiture ?
Oui
Visual Paradigm Online Express Edition
Action Voiture
Action Parking
Choix Voiture
Choix Parking

3.


SA6 Parking Manager
Parking 1
Parking 2
Parking 3
10/10
83.21e
11/15
145.81e
9/30
81.02e

4.


Parking 1 - Conversations
Voiture 1
Voiture 2
Voiture 3
Voiture 4
Voiture 5
Voiture 7
Voiture 10
Voiture 11
Voiture 12
Voiture 13
Voiture 14
Voiture 15
Voiture 16
Voiture 19
Voiture 21
Voiture 23
Voiture 27
Voiture 29
Voiture 29: Est-ce que vous avez de la place ?
Parking 1: Oui
Voiture 29: Tres bien, voici une trame contenant mes informations <29,4,0,0,2>
Parking 1: C'est recu, je peux vous obtenir une place pour 6.632770e.
Voiture 29: Desole, ca ne rentre pas dans mon budget, voici mon offre: 4.823833e
Parking 1: Desole mais je me dois de refuser !
Voiture 29: Nous sommes en semaine, vous pouvez reduire un peu le prix !
Parking 1: C'est vrai mais votre prix est toujours trop haut.
Voiture 29: Je suis un client regulie
Parking 1: Vous etes un client tres fidele mais il nous est impossible d'acceder a votre demande.
Parking 1: Voici une reduction pour vous remercier de votre fidelite: 5.306216
Voiture 29: J'accepte votre offre.
Parking 1: Bienvenue dans mon parking