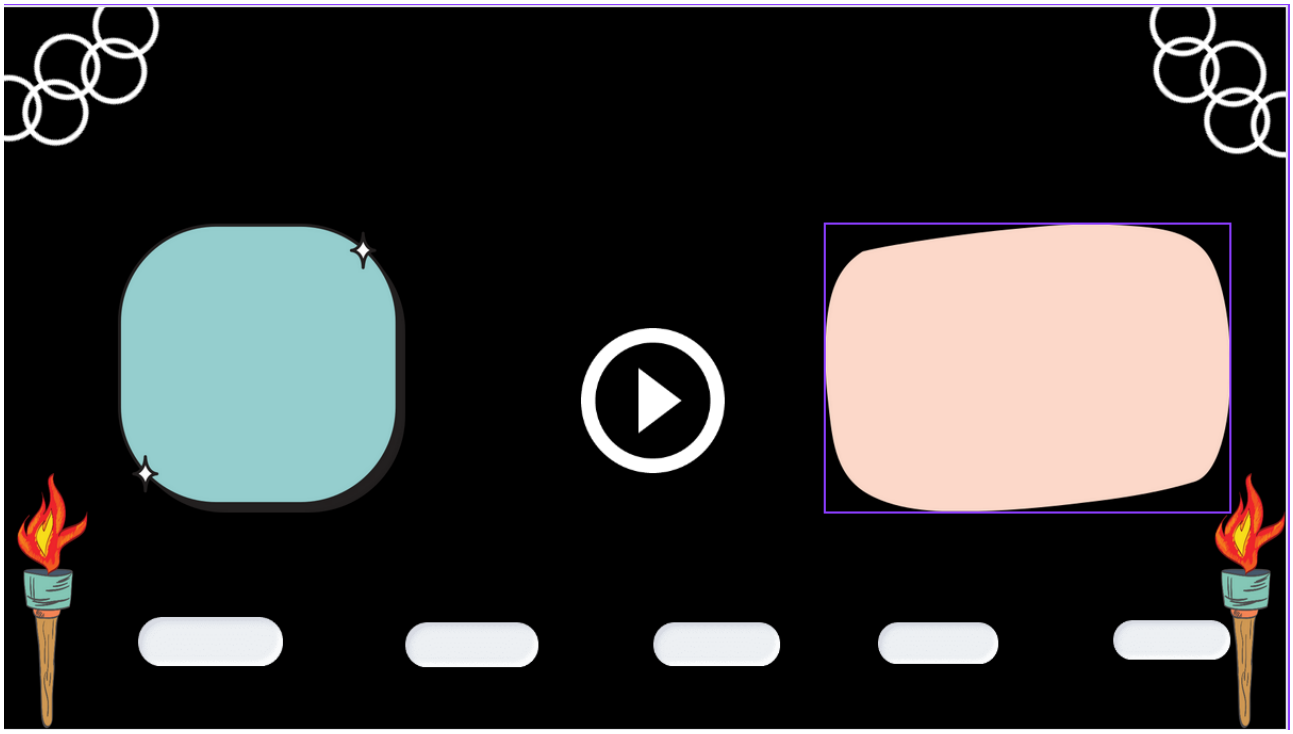
PROJET CPP : OLYMPICS APPLICATION



Equipe :
Linda Tala
Clement Zhuang
MAIN4

Responsable:
Mme. Cécile Braunstein
Encadrant:
M. Johan Hubert

Contents

1 Commande de compilation :

Les compilations du code effectueront automatiquement la génération des pages.

```
//Lancer le code
qmake
make

//nettoyage du code
make clean
```

2 Description de l'application

Notre application, AFFLUX _PARIS, est un programme qui a pour premier objectif de donner une estimation du nombre de personnes présentes à l'intérieur d'un lieu, que ce soit dans un restaurant ou sur une ligne de métro, à un moment précis de la journée. En effet, avec les Jeux olympiques se déroulant dans la ville de Paris, qui compte déjà 2,1 millions d'habitants, cette ville va connaître un afflux d'environ 15,9 millions de touristes. Cette concentration de personnes va avoir un grand impact sur le quotidien des Parisiens, mais également des touristes, notamment sur les transports qui sont déjà un grand sujet de discussion hors période estivale.Cette application est destinée aux personnes qui souhaiteraient avoir une idée de l'affluence d'un lieu afin d'organiser leurs journées sans être beaucoup impactées. L'implémentation que nous avons faite est loin de ce que nous avions prévu de faire au départ, car nous nous sommes éloignés de l'idée de départ, qui était uniquement de faire une sorte de simulation. Plus le projet avançait, plus nous avons trouvé intéressant de créer une application instructive répertoriant différentes informations qu'un touriste pourrait chercher. Ce qui nous a influencés à créer d'autres fonctionnalités pour notre interface. Cette application fait office d'une sorte de guide touristique afin de permettre à ces utilisateurs de découvrir de nouveaux restaurants ou lieux culturels dans la métropole.

2.1 L'implémentation

L'implémentation du projet s'est effectuée à deux niveaux. Le premier niveau était d'implémenter un modèle de simulation de l'affluence, et le deuxième était de construire le guide touristique sur Qt.
En premier lieu, concernant l'implémentation du modèle de simulation, après d'amples recherches, nous nous sommes dirigés vers une simulation par agent. Ce type de simulation permet de reproduire le comportement d'un système, pour nous la ville de Paris, en donnant à chaque entité la capacité de faire leur propre décision et d'interagir avec leur environnement. Chacun des agents de notre système possède des qualificatifs, notamment un budget, un lieu d'appartenance et un temps d'attente. Avec ces informations, nous avons créé plusieurs lieux avec différentes activités comprenant un certain temps pour chacune. L'algorithme, par la suite, détermine le temps d'attente de l'utilisateur en déterminant si le nombre de personnes devant lui est supérieur à la capacité supportée par le lieu.
En deuxième lieu, l'implementation des autres fonctionnalités de l'application. En effet, la simulation est la fonctionnalité principale de notre interface, mais le fait qu'on voulait également faire une sorte de guide afin de conseiller nos utilisateurs dans l'optique où ils seraient coincés dans un arrondissement et que le parcours qu'ils avaient prévu de base ne puisse pas avoir lieu dû à la suraffluence dans une station de métro ou dans un lieu où ils auraient d'autres optiques. Pour cela, nous avons créé différentes touches par rapport aux activités par arrondissement, mais aussi des zones de restriction de Paris lors des Jeux Olympiques. Cette partie est beaucoup plus axée sur l'aspect esthétique de l'application avec Qt.

2.2 Classes utilisées

2.2.1 Classes utilisées pour la simulation de l'affluence

Pour la simulation de l'affluence, nous avons une première classe **Place** qui correspond au lieu associé à l'agent de notre système. Elle a 2 attributs :

- la `Capacité` qui correspond au nombre de personnes pouvant être accueillies dans le lieu en même temps.
- la `Popularité` qui correspond à la popularité du lieu, fait partie d'un des nombreux attributs que nous n'avons pas pu utiliser faute de temps.

Ensuite, pour chaque lieu est associée une **Activité**, ce dernier a pour attribut le:

- le `Temps pris` par chaque activité
- le `Coût associé` qui correspond au prix de chaque activité.

2.2.2 Classes utilisées pour les autres fonctionnalités de l'application

Dans le cadre de notre application destinée à la gestion urbaine et à la simulation de scénarios divers, plusieurs classes ont été développées pour modéliser les interactions et les fonctionnalités requises. Voici une description des principales classes utilisées : **MainWindow** : cette classe sert de fenêtre principale pour l'application. Elle intègre différents modules par l'intermédiaire de widgets et contrôle les interactions principales de l'utilisateur.

- `scene` : un objet de type **MyGraphicsScene** qui gère les éléments graphiques avancés.
- `arrondissements` : instance de la classe **arrondissement**, gérant les spécificités des différents arrondissements.
- `jo` : Classe **JO** qui traite des informations liées aux Jeux Olympiques.
- `ABoutUs` : Gère l'affichage des informations relatives à la section "À Propos".
- `simulation` : Pour la simulation de divers scénarios.

arrondissement : Classe dédiée à la gestion des arrondissements.

- `svgItem` : Utilisé pour l'affichage de cartes ou de plans en format SVG.

AboutUS : Classe permettant de gérer l'interface de la section "À Propos".

- `goToMainWindow` : Méthode pour revenir à la fenêtre principale.

secretgame : Une classe conçue pour ajouter un aspect ludique à l'application, avec des fonctionnalités de jeu cachées.

- `showSecretMessage` : Affiche des messages ou des indices dans le cadre du jeu.

CustomWidget : Cette classe encapsule des widgets personnalisés qui peuvent être réutilisés à travers différentes interfaces de l'application.

- `homeButtonClicked` : Signal émis lors du clic sur le bouton d'accueil.

Chaque classe est spécialement conçue pour interagir de manière cohérente avec les autres, assurant ainsi une navigation fluide et une expérience utilisateur optimale dans l'application de gestion urbaine.

2.3 Étapes de la simulation

2.3.1 Initialisation du modèle

En premier lieu, nous allons initialiser le modèle CityModel qui sera la pièce centrale chargée de gérer la simulation. Ce dernier va contenir 3 attributs: Place, Person et enfin un attribut pour avoir le temps de la simulation.

2.3.2 Initialisation des personnes

Les personnes sont initialisées de manière aléatoire et leur nombre est donné en entrée par l'utilisateur avec un budget compris entre 1000€ et 5000€ et un currentLocation initialisé à null.Ensuite, on va faire avancer la simulation par pas de 1, qui correspond à 30 minutes, afin d'essayer de bouger la personne. Si son waitingTime n'est pas 0 alors il ne bougera pas. En effet, nous avons posé l'hypothèse que les personnes sont sûres de leurs actions.Nous aurons un tableau pour suivre l'avancement de la simulation par rapport aux différents lieux.

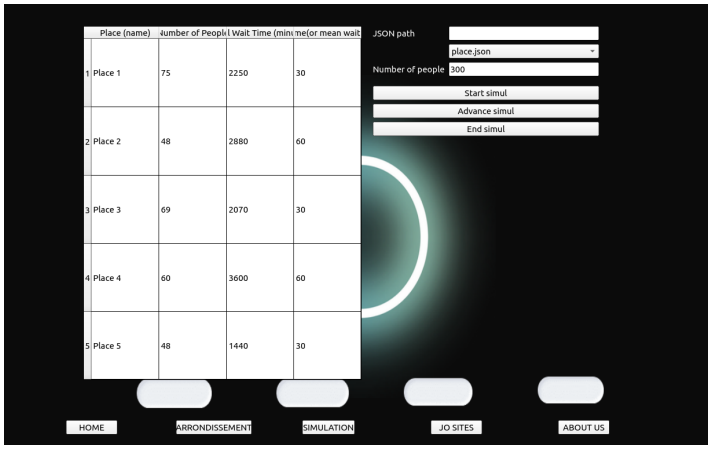


Figure 1: Page de simulation avec modèle initialisé

2.4 Les axes d'amélioration

L'implémentation actuelle peut être améliorée en se concentrant sur la simulation dynamique de l'affluence pendant les JO. Des ajustements au modèle par agent et une gestion plus avancée des données auraient été nécessaires pour atteindre ces objectifs.Pour la simulation, plusieurs améliorations sont envisageables. Pour commencer, dans notre simulation, nous considérons seulement le temps d'attente, qui est un cumul entre le temps estimé dans la file d'attente pour un lieu, si une queue existe, ajoutez à cela le temps pris par l'activité. Nous pourrions donc ajouter un attribut qui prendrait seulement le waitingTime ,temps d'attente, et un autre timeBeforeLeave, qui correspond au temps avant qu'il ne parte.Ensuite, il serait intéressant d'ajouter dans la simulation un modèle qui simule l'arrivée graduelle de personnes dans Paris tout au long de la durée des Jeux Olympiques. Dans notre simulation actuelle, nous n'avons qu'une initialisation de population au début.Un des problèmes que nous avons également remarqué dans la simulation était que lorsque Nous avons quelque problème en terme de type de données. En effet, quand nous initialisions un trop grand nombre de personnes et que nous voulions suivre l'attente cumulée sur un endroit, la valeur devenait trop grande pour un int .

3 Interface graphique et autres fonctionnalités

Notre but étant de créer une application pour simuler une affluence, nous nous sommes également mis à la place de notre public visé afin de savoir quels autres types d'informations leur seraient utiles à savoir et sous quels formats. L'application développée est une application graphique utilisant la bibliothèque Qt. Elle présente une interface utilisateur lui permettant d'explorer différentes fonctionnalités liées à la ville de Paris. Les principales caractéristiques de l'application incluent la visualisation de la carte de Paris, l'accès aux informations sur les arrondissements, les zones restrictives pendant les Jeux Olympiques et les épreuves.

3.1 Page Home

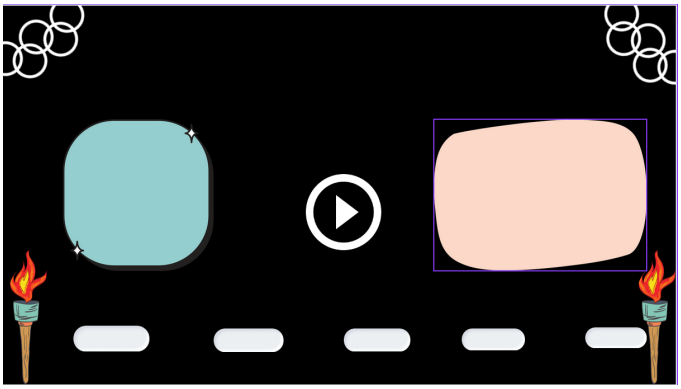


Figure 2: Home Page

La classe principale, `MainWindow`, qui organise les différentes fonctionnalités et intègre des éléments graphiques tels que des images et des vidéos pour offrir une expérience immersive.

La classe `MainWindow` gère une variété de fonctionnalités, dont certaines sont déclenchées par des boutons sur l'interface. Par exemple, le bouton "Play Video" active la lecture d'une vidéo présentant des aspects spécifiques de Paris. Le bouton "Arrondissement" conduit à une autre page, représentée par la classe `arrondissement`, qui affiche des informations détaillées sur les différents arrondissements de Paris. Le système de scènes et d'items graphiques, illustré par la classe `MyGraphicsScene`, permet

une représentation visuelle de la carte de Paris à travers un fichier SVG.

Par contre, dû à un soucis technique, lorsqu'on lance la vidéo, la fenêtre crash après, donc on vous conseille de jouer la vidéo à la toute fin après découverte totale de l'application.

3.2 Page Arrondissement

Dans la classe Arrondissement, nous avons créé une interface utilisateur centrée sur la présentation des différents arrondissements de Paris. Cette classe hérite de QMainWindow, ce qui lui confère des fonctionnalités avancées pour gérer l'interface utilisateur.

La structure de la classe comprend une variété de widgets graphiques, tels que des boutons QPushButton qui déclenchent des actions spécifiques lorsqu'ils sont cliqués. Ces actions incluent la navigation vers d'autres pages, comme la page JO, Jeux Olympiques, la page de simulation , simul, et la page AboutUs qui fournit des informations supplémentaires sur l'application. Les boutons spécifiques aux arrondissements déclenchent des messages d'information, QMessageBox, contenant des détails sur chaque arrondissement, y compris une description et des emojis représentant les caractéristiques emblématiques. Ces boutons montrent également la diversité culturelle, gastronomique et récréative présente dans chaque quartier.

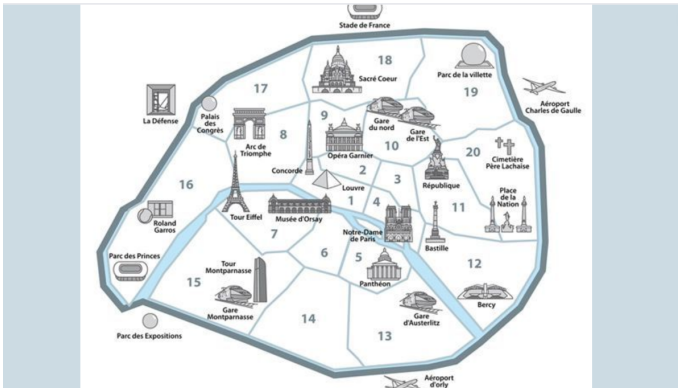


Figure 3: Arrondissements Page

3.3 Page Jeux Olympiques

La classe JO représente une fenêtre de l'application qui fournit des informations sur les Jeux Olympiques (JO) à Paris. Comme dans la classe Arrondissement, elle hérite de la classe QMainWindow de Qt, ce qui lui donne des fonctionnalités étendues pour gérer l'interface utilisateur. Elle utilise également des éléments graphiques tels que `QGraphicsScene` et `QGraphicsSvgItem` pour afficher des images et des scènes liées aux JO. Par exemple, elle affiche une carte des sites olympiques et une carte des zones qui seront restrictives pendant les JO. La fonction `afficherInformationsZone` est appelée lorsque l'utilisateur clique sur l'un des boutons représentant les différentes zones (rouges, bleues, grises) dans la carte des zones colorées. Cette fonction affiche une boîte de message (`QMessageBox`) avec des informations spécifiques sur la zone sélectionnée.

3.4 Page About US

La classe AboutUS représente une fenêtre de l'application qui fournit des informations sur l'équipe ou le sujet associé à l'application. Elle hérite également de la classe QMainWindow de Qt, ce qui lui donne des fonctionnalités étendues pour gérer l'interface utilisateur.La classe contient des slots pour gérer les actions associées à différents boutons, tels que le retour à la page principale (goToMainWindow), la navigation vers d'autres pages (on-JOButton-clicked, on-arrondissementButton-clicked, on-simButton-clicked) et l'ouverture d'un jeu secret (on-gameButton-clicked).

3.5 Page Secret Game

Afin de rendre notre application un peu plus intéressante, nous avons décidé de mettre en place un jeu. La fenêtre Secret Game représente le lien entre les deux projets ou la transition. On passe d'un projet de simulation au projet de jeu vidéo afin de permettre à l'utilisateur d'avoir une application ludique.
La classe secretgame représente une fenêtre secrète de notre application qui peut être accédée depuis une autre partie de l'application, l'About US page. Un bouton nommé onGameButton est connecté à la fonction Vous pourrez aussi le trouver sur le git avec le fichier xml utilisé pour umbrello.

Le jeu a été retrouvé à ce lien git : $https : //github.com/Przemekkkth/Mario\_Qt - Cpp$.
Lorsque onGameButton est cliqué, la fonction onButtonClicked est appelée, qui lance un jeu secret en exécutant un programme externe (MarioQt) à l'aide de **QProcess : :startDetached**.

3.6 Page Mario Bros

Une idée pour améliorer notre application était d'y incorporer un jeu, car nous étions dans la pensée que si l'utilisateur n'est pas présent dans une zone avec une connexion sans fil fiable , dans le métro par exemple, il aurait au moins de quoi se changer les idées.Nous avons réussi à trouver sur un repositiry GIT sous la licence MIT nous permettant d'utiliser un jeu Mario Bros que nous avons donc connecté à notre interface Qt. Le jeu a été retrouvé à ce lien git : $https : //github.com/Przemekkkth/Mario_Qt - Cpp$.

4 Utilisation des contraintes

4.1 Rapport

Voici le diagramme UML de notre simulation. Vous pourrez aussi le trouver sur le Git avec le fichier XML utilisé pour

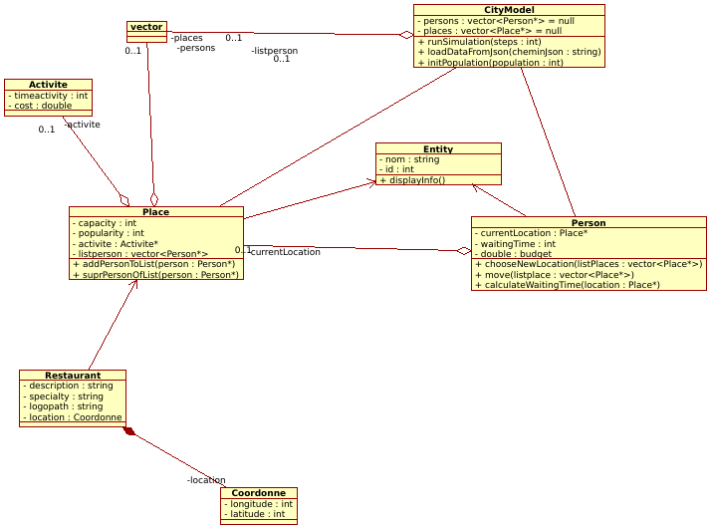


Figure 4: Diagramme de classe pour la simulation

Umbrello.
En terme d'implémentation, la partie dont je (Clément) suis le plus fier est d'avoir pu gérer toutes les dépendances de la simulation et de faire en sorte qu'elles fonctionnent, surtout avec l'initialisation de Place qui me paraissait compliquée à implémenter, surtout par le biais de fichiers json.
Ce dont je (Linda) suis la plus fière est la gestion des différentes fenêtres qui est assez rapide, mais également le fait d'avoir pu lier 2 projets différents ensemble en changeant quelques paramètres afin de pouvoir installer le jeu sur notre application.
La plus grande fierté de notre groupe est d'avoir manié un nouvel outil, Qt, et d'avoir à la fin un résultat qui répond à la majorité de nos attentes, malgré les différents problèmes qu'on a eu au début de notre projet du fait que nous n'utilisions pas la même version de Qt et que certaines fonctions ne sont pas appropriées.

4.2 Simulation par agent

L'utilisation des contraintes dans l'application se manifeste à travers le modèle par agent, où chaque entité suit des règles spécifiques basées sur son budget, son lieu d'appartenance et son temps d'attente.

4.3 Classe et méthode pour la simulation

Nous avons implémenté plusieurs classes avec des dépendances entre elles. Pour commencer, nous avons une classe Activity qui possède des dérivées, DiningActivity ou encore SightseeingActivity. Nous avons défini dans Activity une méthode virtuelle pour l'affichage de données relatives à l'activité, sachant que Activity est une classe virtuelle.Pour ce qui est des 3 niveaux de hiérarchie, cela s'est fait avec les classes Restaurant qui héritaient de Place et ce dernier héritant d'Entity. Un tel choix vient de la volonté d'avoir une instance unique de la classe Place, d'où l'héritage de la classe Entity. Pour ce qui est des dérivées de la classe Place, nous avions l'intention de créer des endroits différents et donc aux méthodes et attributs différents, ce qui amena à l'héritage.

5 Procédure d'Installation et Exécution

- **Installation des Bibliothèques Qt :** Avant de compiler l'application, assurez-vous que les bibliothèques Qt soit installées sur votre système.

  Pour Linux:

```
    \\Outils de developpement
        sudo apt-get update
        sudo apt-get install build-essential

    \\Installation de Qt
    sudo apt-get install qt5-default

    \\Verifier l'installation
    qmake --version
```

Pour macOS:

```
        \\installer Homebrew
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
   HEAD/install.sh)"

\\Installer Qt via Homebrew :
brew install qt
\\Verifier l'installation :
qmake --version
```

Pour Windows:

- Télécharger Qt sur leur site
- Sélectionner téléchargement OpenSource

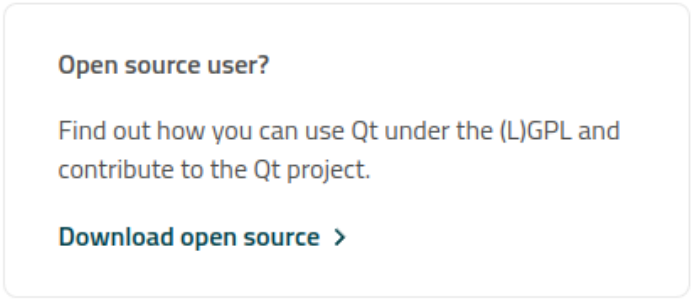


Figure 5: Téléchargement open source

- Suivre instructions de l'installateur en se rassurant d'avoir les bibliothèques comme MSVC ou MinGW incluses
- Ajoutez le répertoire bin de Qt à votre variable d'environnement PATH. Enfin exécutez:

```
    qmake --version
```

- **Compilation du Code :** Utilisez un environnement de développement compatible avec Qt, tel que Qt Creator. Chargez le projet et compilez-le en suivant les instructions spécifiques à votre environnement.
- **Exécution de l'application** : Lancez l'application compilée. Vous pouvez explorer la carte de Paris, accéder aux informations sur les arrondissements, simuler des événements, et plus encore.

6 Conclusion

Notre application, $AFFLUX_PARIS$, permet en temps réel d'observer l'occupation d'un lieu en faisant une simulation de l'affluence des personnes à l'intérieur. Elle constitue une outil de gestion de temps mais également outil touristique pour les utilisateurs pendant la période des Jeux Olympiques.

Elle peut être utilisée en dehors de cette période permettant ainsi aux utilisateurs de la ville de Paris de mieux organiser leur temps de trajet tout en profitant des ressources que cette ville a à offrir.

7 Bibliographie

Lien GIT vers le projet : https://github.com/LindaMAIN/CPP-PROJECT

8 Annexes

https://www.youtube.com/watch?v=yQ2As3PSJHM
https://www.youtube.com/watch?v=DchGfN$_F udQt = 883s$
https://www.youtube.com/watch?v=-Gqx33rvKz0
https://www.youtube.com/watch?v=nWxTlBTaav8t=16s
https://www.youtube.com/watch?v=6$_e lY8O20I8$
https://www.youtube.com/watch?v=tjuBxjmhGLs
https://www.youtube.com/watch?v=5JVLO8yBMXAt=1s
https://github.com/Przemekkkth/Mario$_Q t - Cpp$