Lyon Poussette GPS : Schraen Matthias / Dufour Tim / Coron Valentin / Donadey Remi

Découpage Fonctionnel

Fichier principal (Projet_Poussette.py)

Fenêtre principale

classe Mainwindow() :

méthodes de classes :

-init(self) → Fonction constructeur appelée lorsque l'on crée une instance de notre application

-initWidget_load(self) → Initialise les différents widgets, la taille et le titre de la fenêtre de chargement.

- initWidget_main(self) → Intialise les widgets des différentes frames de l'interface graphique, même celles qui seront affichées après (par un evénement utilisateur).

- loop(self) → lance la boucle principale d'attente des événements de la fenêtre

- lancement_auto(self,event) → Fonction callback du bouton de lecture automatique Déclenche le défilement automatique des étapes du parcours (carrefours successifs) sur la fenêtre trajet. Appelle : automatique,init_widget_parcour,show_iti

-maj_auto(self,val_vitesse) → Fonction pseudo-callback du slider de selection de la vitesse de lecture Calcul le délai écoulé depuis l'affichage automatique de la dernière étape et passe à l'étape suivante si ce delai est plus long que celui lié à la nouvelle vitesse selectionnée par le slider.

-automatique(self) → Gère le défilement automatique des étapes du parcours (carrefours successifs) sur la fenêtre trajet Efface le carrefour dessiné et retrace automatiquement le carrefour suivant sur la fenêtre trajet puis s'appelle de nouveau après un delai lié à la vitesse définie par le slider. Appelle : show_iti

-get_entry_start(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse de départ (fonction déclenchée avec n'importe quelle touche du clavier dans la combobox de départ) Découpe la saisie en éléments d'adresse (numéro, rue, ...), identifie une liste d'adresses correspondantes, et met à jour les propositions de la combobox avec cette liste. Appelle : give_troncon_address,gestion_saisie

-down_start(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse de départ (fonction déclenchée avec la touche Return du clavier dans la combobox de départ) Met à jour la liste de propositions de la combobox de la même façon que get_entry_start(self, event) et descend la liste de la combobox. Appelle : give_troncon_address,gestion_saisie

-choose_start(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse de départ (fonction déclenchée par la selection d'une proposition dans la liste de la combobox de départ) Redécoupe la valeur du champs de la combobox en éléments d'adresse (numéro, rue, ...) et identifie

l'adresse correspondante pour trouver ses coordonnées GP puis appelle l'identification du tuple (code_fuv,code_troncon) avec ces coordonnées GPS.

-effacer_start(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse de départ (fonction déclenchée par le clic du bouton gauche sur la combobox de départ) Efface le texte indicatif "Départ" pour permettre la saisie de l'utilisateur.

-ecrire_start(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse de départ (fonction déclenchée par la sortie du "focus" de la combobox de départ) Remet le texte indicatif "Départ" en l'absence de saisie utilisateur.

-get_entry_end(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse d'arrivée (fonction déclenchée avec n'importe quelle touche du clavier dans la combobox d'arrivée') Découpe la saisie en éléments d'adresse (numéro, rue, ...), identifie une liste d'adresses correspondantes, et met à jour les propositions de la combobox avec cette liste.

--down_end(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse d'arrivée (fonction déclenchée avec la touche Return du clavier dans la combobox de d'arrivée) Met à jour la liste de propositions de la combobox de la même façon que get_entry_end(self, event) et descend la liste de la combobox. Appelle : give_troncon_address,gestion_saisie

-choose_end(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse de départ (fonction déclenchée par la selection d'une proposition dans la liste de la combobox de départ) Redécoupe la valeur du champs de la combobox en éléments d'adresse (numéro, rue, ...) et identifie l'adresse correspondante pour trouver ses coordonnées GPS puis appelle l'identification du tuple (code_fuv,code_troncon) avec ces coordonnées GPS.

-effacer_end(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse d'arrivée (fonction déclenchée par le clic du bouton gauche sur la combobox d'arrivée) Efface le texte indicatif "Arrivée" pour permettre la saisie de l'utilisateur.

-ecrire_end(self,event) → Fonction callback sur le champs d'entrée associé à l'adresse d'arrivée (fonction déclenchée par la sortie du "focus" de la combobox d'arrivée) Remet le texte indicatif "Arrivée" en l'absence de saisie utilisateur.

-recup_fuv_troncon(self) → Identifie en fonction des coordonnées GPS le tuple (code_fuv,code_troncon) de départ et celui d'arrivée Ces tuples indentifient les troncons de manière unique. Appelle : give_troncon_nearest_gps

-start_research(self,event) → Fonction callback du bouton de calcul de l'itinéraire. Lance la recherche de l'itinéraire en fonction du mode de recherche choisi par l'utilisateur et génère le changement de frame, le calcul et l'affichage des étapes, et l'affichage de l'itinéraire sur la carte si un trajet est trouvé Retourne à l'utilisateur un message d'erreur sinon. Appelle : show_large_map,consigne_noeud,open_window_trajet_middle

-show_large_map(self) → Affiche le trajet à affectuer et les marqueurs de départ et d'arrivée sur l'interface graphique.

-bouton_change_iti(self,event) → Fonction callback du bouton de changement d'itinéraire Demande confirmation à l'utilisateur et repasse à la frame d'acceuil dans le cas positif Supprime aussi dans ce cas les éléments associé au trajet en cours (tracé sur la carte, fenêtre trajet, ...).

-open_window_trajet(self,event) → Fonction callback du bouton commencer le trajet Ouvre la fenetre trajet qui affiche l'itinéraire carrefour par carrefour Affiche le premier carrefour. Appelle : init_widget_parcour,show_iti

-open_window_trajet_middle(self,idx) → Fonction pseudo-callback des boutons étapes de l'itinéraire Ouvre la fenetre trajet qui affiche l'itinéraire carrefour par carrefour Affiche le carrefour rattaché au bouton cliqué. Appelle : init_widget_parcour,show_iti

-load_all_datas(self) → Effectue le chargement des données, met à jour la barre de chargement puis appelle l'affichage principal (fonction déclenchée après l'affichage de la fenêtre de chargement). Appelle : charger_donnees_troncon,charger_donnees_troncon,correction_dico_noeuds,charger_donne_adj,charger_donnees_adj_poussette,charger_donnees_adj_velo,charger_donnees_adj_voiture,charger_donnees_adj_pied,charger_donnees_adresses,charger_donnees_centre.

Fenêtre Trajet

- initWidget_parcour(self) → initialise l'ensemble des widgets sur la fenêtre et les place.

-precedent(self,event) → Fonction callback du bouton précédent Efface le carrefour dessiné et retrace le carrefour précédent sur la fenêtre trajet. Appelle : show_iti

-suivant(self,event) → Fonction callback du bouton suivant Efface le carrefour dessiné et retrace le carrefour suivant sur la fenêtre trajet et ferme la fenêtre trajet si on est au dela de la dernière étape. Appelle : show_iti

- show_iti(self, fuv_tr_pre, fuv_tr_suiv) → Affiche dans la fenêtre trajet, l'itineraire à suivre au carrefour renseigné et centre l'affichage de la carte de la fenêtre principale sur ce carrefour. Appelle : calcul_dist_min,compute_cross,calcul_angle,rotation_repere,distance,norme_min,xy_cartesien,dessine_noeud,instructions

-dessine_noeud(self, dico_fuv_tr_carte, fuv_tr_pre, fuv_tr_suiv, co_nord, cote_echelle, echelle_choisie) → Dessine l'intersection renseignée avec la route à suivre, les indications, une boussole et une echelle.

Fichier secondaire

Load_Files.py → fichier avec les méthodes pour charger les fichiers contenant les données relatives aux routes et chemins ainsi que les différents algorithmes de calcul pour trouver le chemin le plus court ou afficher la carte et le trajet fidèlement à la réalité.

Méthodes :

- charger_donnees_troncon() → Ouvre le dictionnaire de données du Grand Lyon relatif aux troncons et utilise les données pour créer un dictionnaires des troncons avec leurs caractéristiques et un dictionnaires des noeuds .

-charger_donnees_chaussees(dico_noeuds,dico_rues) → Ouvre le dictionnaire de données du Grand Lyon relatif aux chaussées et trotoirs et complète les dictionnaires créés par charger_donnees_troncon() avec les données du fichier ouvert.

-correction_dico_noeuds(dico_noeuds) → Ajuste le dictionnaire des noeuds pour enlever les noeuds qui ne sont pas des carrefours (noeuds qui sont reliés à moins de 2 troncons).

-charger_donnees_adj(dico_noeuds) → Creation du dictionnaire d'adjacence

-charger_donnees_adj_poussette(dico_noeuds,dico_rues) → Crée le dictionnaire d'adjacence des troncons entre eux en excluant les troncons qui ne sont pas compatibles avec les critères définis pour le moyen de transport poussette.

-charger_donnees_adj_velo(dico_noeuds,dico_rues) → Crée le dictionnaire d'adjacence des troncons entre eux en excluant les troncons qui ne sont pas compatibles avec les critères définis pour le moyen de transport vélo.

-charger_donnees_adj_voiture(dico_noeuds,dico_rues) → Crée le dictionnaire d'adjacence des troncons entre eux en excluant les troncons qui ne sont pas compatibles avec les critères définis pour le moyen de transport voiture.

-charger_donnees_adj_pied(dico_noeuds,dico_rues) → Crée le dictionnaire d'adjacence des troncons entre eux en excluant les troncons qui ne sont pas compatibles avec les critères définis pour le moyen de transport à pied.

-charger_donnees_adresses() → Ouvre le dictionnaire de données du Grand Lyon relatif aux points de débouché (adresses existantes) et utilise les données pour créer des dictionnaires des adresses existantes avec leurs coordonnées GPS, rangées respectivement par numéro puis voie puis commune, voie puis numéro puis commune, ou commune puis voie puis numéro.

-charger_donnees_centre(dico_adresses_communes) → Calcule les coordonnées GPS des centres des villes/communes et enregistre ces coordonnées dans le dictionnaire des adresses rangées par commune puis voie puis numéro avec pour nom de voie "centre" et numéro de voie "0".

-gestion_saisie(saisie_user,l_communes) → Découpe la saisie de l'utilisateur et identifie des éléments d'adresse (numéro, rue, ...) Retourne ces éléments ou None pour les élément(s) non-identifié(s). Retourne : l'ensemble des informations sur la saisie de l'utilisateur (numéro,rue,communes,co_gps)

-dist_lat_long_deg(start_lat,start_lon,end_lat,end_lon) → Retourne la distance en mètres entre deux points sur la Terre à partir de leurs coordonnées GPS en degrés. Retourne : la distance a vol d'oiseau entre ces 2 points

-a_star(start, goal, rues_adjacentes, dico_rues, crit_sens, crit_vitesse) → Algorithme de parcours du graphes des troncons qui donne le chemin le plus direct et le plus court entre deux troncons Ce

parcours de graphe peut être orienté ou non en fonction de la valeur du booléen crit_sens, et est pondéré soit par la distance, soit par le temps de trajet (relatif à la vitesse) en fonction de la valeur du booléen crit_vitesse. Appelle : dist_lat_long_deg. Retourne : le chemin calculé ainsi que la distance de ce chemin.

- compute_iti(adjacence_rues, depart_fuv_troncon, arrivee_fuv_troncon) → algorithme qui va calculer le chemin le plus court entre un départ et une arrivée dans un dictionnaire d'adjacence préalablement modifié pour contenir seulement les tronçons qui respectent les conditions de l'utilisateur.
Appelle : algo_a_star, dist_between_points, setup_adjacence_param
Retourne : l'itinéraire trouvé

- compute_cross(fuv_tr_pre, fuv_tr_suiv, dico_rues, rues_adjacentes) → Identifie le carrefour associé aux identifiants des troncons précédents et suivants, identifie les troncons adjacents à ce carrefour, convertit les coordonnées GPS de tous ces troncons en coordonnées cartésiennes sans normalisation et range toutes ces informations dans un dictionnaire retourné par la fonction en plus de la latitude et la longitude du noeud. Appelle : xy_lat_long. Retourne : le dico des coordonnées cartesiennes ainsi que les coordonnées gps du noeud

- algo_a_star(start_fuv_troncon, end_fuv_troncon, adj_rue) → on va utiliser l'algorithme A* pour trouver le chemin le plus court entre le point de départ et le point d'arrivé de l'utilisateur. En effet, il semble plus optimisé dans notre cas de graphe « spatial ». En effet on connaît facilement une direction optimale à suivre, la distance en ligne droite « à vol d'oiseau » entre le départ et l'arrivée.
Retourne : la liste des codes fuv+tronçon emprunté par l'itinéraire calculé

- dist_between_points(start_fuv_troncon, end_fuv_troncon, adj_rue) → permet de retourner la distance entre ces 2 points « à vol d'oiseau ».
Retourne : la distance « à vol d'oiseau »

- calcul_min_max_xy(dico_sommets) → détermine les coordonnées min et max en x et y des points dans le plan du canevas
Appelle : xy_from_lat_long
Retourne : les x et y min et max

- xy_lat_long(latitude, longitude,latitude_ref) → effectue la conversion de la latitude et longitude en coordonnées x, y dans le plan du canevas sans normalisation
Retourne : les coordonnées x et y calculées en fonction de la latitude et longitude mis en paramètre

-calcul_angle(x1,y1,x2,y2) → Retourne l'angle entre un segment (défini par deux points en coordonnées cartésiennes) et un axe horizontal orienté de gauche à droite (l'axe x de l'interface Tkinter) La valeur de l'angle est comprise entre -pi/2 et 3pi/2. Retourne : l'angle de rotation à effectuer.

-rotation_repere(angle,dico_fuv_tr_adj) ->Retourne le dictionnaire avec les troncons précédent, suivant et adjacents et leurs coordonnées cartésiennes sans normalisation mais après rotation du

repère pour que le troncon précédent soit vertical ascendant. Retourne : le nouveau dictionnaire des code fuv avec sa rotation.

- xy_cartesien (dist_min, dico_fuv_tr_rot, xy_noeud, width_canvas, height_canvas) → Retourne le dictionnaire avec les troncons précédent, suivant et adjacents et leurs coordonnées cartésiennes après normalisation de ces coordonnées à l'echelle de l'affichage
Retourne : le nouveau dictionnaire avec les coordonnées dans le systeme cartesien

-calcul_norme_min(dico_fuv_tr_rot) → Retourne la norme infinie minimale entre le noeud et les extrémités des troncons du dictionnaire. Retourne: cette norme

-calcul_dist_min(dico_fuv_tr_adj) → Retourne la norme deux (norme canonique) minimale entre le noeud et les extrémités des troncons du dictionnaire. Retourne: la distance minimale

-distance(co_gps, index1, index2) → Retourne la distance selon la norme deux entre deux points définis par leurs index dans une liste de coordonnées.

-instructions(dico_fuv_tr_carte, fuv_tr_pre, fuv_tr_suiv, dico_rues) → Retourne les instructions en texte selon la nouvelle direction et route à prendre pour un carrefour donné. Appelle : distance,calcul_angle. Retourne le texte d'instruction a afficher dans la fenetre trajet

-consigne_noeud(fuv_tr_pre, fuv_tr_suiv, dico_rues, rues_adjacentes) → Identifie le carrefour associé aux identifiants des troncons précédents et suivants, identifie les troncons adjacents à ce carrefour, effectue la rotation et la normalistion du repère, et retourne les instructions pour ce carrefour. Appelle : compute_cross, calcul_dist_min, distance, alcul_angle, rotation_repere, calcul_norme_min, xy_cartesien,instruction. Retourne: le texte des instruction afficher

- give_troncon_nearest_address(numero, rue, commune, com_possibles, latitude, longitude, dico_adresses_num, dico_adresses_rues, dico_adresses_communes) → Retourne une liste (au maximum 6 éléments) des adresses possibles en fonction des éléments d'adresse (complets ou incomplets) identifiés dans la saisie utilisateur. Retourne : une liste d'addresse possible

- give_troncon_nearest_gps(co_gps_start, co_gps_end ,dico_rues,choix_transport) → Trouve le couple (fuv,troncon), présent dans le dictionnaires des troncons, le plus proche des coordonnées GPS de départ renseignées et celui le plus proche des coordonnées GPS d'arrivée également renseignées Ces couples sont choisis en respectant les critères imposés par le choix du mode de transport de l'utilisateur.
Retourne : le code troncon+FUV (dictionnaire d'adjacence) associé aux coordonnées GPS pour le départ et l'arrivée. Appelle : dist_lat_lon_deg