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 de la fenêtre de chargement

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

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

- lancement_auto(self,event) → fonction callback qui lance le défilement automatique du parcours sur l'interface détails. Appelle : automatique,init_widget_parcour,show_iti

-maj_auto(self,val_vitesse) → Permet de mettre a jour la vitesse de défilement automatique des carrefour à chaque fois que l'utilisateur change de valeur de cette vitesse de défilement sur le slidder.

-automatique(self) → Efface et retrace automatiquement le carrefour suivant sur la fenêtre détail du trajet. Appelle : show_iti

-get_entry_start(self,event) → Fonction callback sur le champs d'entrée associée a l'adresse de départ. Appelle : give_troncon_address,gestion_saisie

-down_start(self,event) → Fonction callback appellee lorsque l'on cherche à dérouler la combobox pour proposer les choix de l'adresse de départ. Appelle : give_troncon_address,gestion_saisie

-choose_start(self,event) → Fonction callback appelle lorsque l'on clique sur un choix proposée dans la combobox de départ.

-effacer_start(self,event) → Fonction callback appelle lorsque l'on cherche à écrire dans le champ d'entrée de l'adresse de départ pour effacer le choix précédent

-ecrire_start(self,event) → fonction callback pour écrire dans le champ d'entrée de l'adresse de départ l'adresse choisie.

-get_entry_end(self,event) → Fonction callback sur le champs d'entrée associée a l'adresse d'arrivée. Appelle : give_troncon_address,gestion_saisie

--down_end(self,event) → Fonction callback appelle lorsque l'on cherche à dérouler la combobox pour proposer les choix de l'adresse d'arrivée. Appelle : give_troncon_address,gestion_saisie

-choose_end(self,event) → Fonction callback appelle lorsque l'on clique sur un choix proposée dans la combobox d'arrivée.

-effacer_end(self,event) → Fonction callback appelle lorsque l'on cherche à écrire dans le champ d'entrée de l'adresse d'arrivée pour effacer le choix précédent

-ecrire_end(self,event) → fonction callback pour écrire dans le champ d'entrée de l'adresse d'arrivée l'adresse choisie.

-recup_fuv_troncon(self) → Récupère le code (fuv,troncon) en fonction des coordonnées gps du point actuel. Appelle : give_troncon_nearest_gps

-start_research(self,event) → Fonction callback du bouton de calcul de l'itinéraire, il lance la recherche de l'itinéraire. Appelle : show_large_map,consigne_noeud,open_window_trajet_middle

-show_large_map(self) → Affiche la carte principale et le trajet à effectuer sur l'interface graphique

-bouton_change_iti(self,event) → Fonction callback appelée lorsque l'on clique sur le bouton pour changer d'itinéraire.

-open_window_trajet(self,event) → Fonction callback du bouton commencer le trajet, ouvre la fenêtre du trajet carrefour par carrefour. Appelle : init_widget_parcour,show_iti

-open_window_trajet_middle(self,idx) → Fonction callback du bouton commencer le trajet, ouvre la fenetre du trajet carrefour par carrefour. Appelle : init_widget_parcour,show_iti

-load_all_datas(self) → lance le chargement des données pendant la fenêtre initiale 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

- start_research(self, event) → fonction callback du bouton « calcul itinéraire ». Lance le calcul de l'itinéraire, change les widgets de la fenêtre principale.
Appelle : setup_adjacence_param, compute_iti, draw_iti_gene

Fenêtre Trajet

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

-precedent(self,event) → Fonction callback appelée lorsque l'on clique sur le bouton précédent. Appelle : show_iti

-suivant(self,event) → Fonction callback appelée par un clic sur le bouton suivant. Appelle : show_iti

- show_iti(self, fuv_tr_pre, fuv_tr_suiv) → Affiche dans cette fenêtre trajet, l'itinéraire a suivre au carrefour actuel. 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) → Agence les différentes formes pour dessiner l'intersection actuelle avec la route à suivre.

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() → Charge des dictionnaires les données du fichier relatif au troncons.

-charger_donnees_chaussees(dico_noeuds,dico_rues) → Ajoute les données relatives au fichier chaussee et trotoirs.

-correction_dico_noeuds(dico_noeuds) → Ajuste le dictionnaire des noeuds pour enlever les doublons inutiles

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

-charger_donnees_adj_poussette(dico_noeuds,dico_rues) → creer le dictionnaire d'adjacence pour le moyen de transport poussette.

-charger_donnees_adj_velo(dico_noeuds,dico_rues) → creer le dictionnaire d'adjacence pour le moyen de transport velo.

-charger_donnees_adj_voiture(dico_noeuds,dico_rues) → creer le dictionnaire d'adjacence pour le moyen de transport voiture.

-charger_donnees_adj_pied(dico_noeuds,dico_rues) → creer le dictionnaire d'adjacence pour le moyen de transport pied choisi par l'utilisateur.

-charger_donnees_adresses() → charge les donnees des debouches (lieu d'arrivee et depart)

-charger_donnees_centre(dico_adresses_communes) → calcule les coordonnees gps des centres des villes/communes

-gestion_saisie(saisie_user,l_communes) → Donne l'adresse complete valide d'une saisie de l'utilisateur. 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) → Donne la distance entre 2 points connaissant leurs coordonnées GPS. 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 de graphs qui donne le chemins le plus courts entre 2 points en suivant le graphs. 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) → Permet de convertir les coordonnées GPS (lat,long) en un systeme de coordonnées cartesiennes pour tracer facilement les routes sur un canvas. 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) → Donne l'angle entre 2 points dans le systeme de coordonnees GPS.
Retourne : l'angle de rotation à effectuer.

-rotation_repere(angle,dico_fuv_tr_adj) ->Effectue la rotations des coordonnées pour changer de repere dans les coordonnées cartesienne (pour toujours mettre la route suivante vers le haut même vision que dans la voiture). Retourne : le nouveau dictionnaire des code fuv avec sa rotation.

- xy_cartesien (dist_min, dico_fuv_tr_rot, xy_noeud, width_canvas, height_canvas) → donne un dictionnaire en coordonnée cartesienne et a l'echelle de l'affichage
Retourne : le nouveau dictionnaire avec les coordonnées dans le systeme cartesien

-calcul_norme_min(dico_fuv_tr_rot) → donne la norme minimale. Retourne: cette norme

-calcul_dist_min(dico_fuv_tr_adj) → Calcule la distance minimale. Retourne: la distance minimale

-distance(co_gps, index1, index2) → calcule la distance entre 2 points.

-instructions(dico_fuv_tr_carte, fuv_tr_pre, fuv_tr_suiv, dico_rues) → Retourne les instructions en texte selon la nouvelle direction/route à prendre. 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) → Donne la consigne associe a un noeud pour un suivant. 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) → Donne une liste des adresse possible en rapport avec la saisie de l'utilisateur. Retourne : une liste d'addresse possible

- give_troncon_nearest_gps(co_gps_start, co_gps_end ,dico_rues,choix_transport) → permet de trouver le code tronçon+FUV le plus proche des coordonnées GPS données par l'utilisateur pour le départ et l'arrivée.
Retourne : le code troncon+FUV (dictionnaire d'adjacence) associé aux coordonnées GPS pour le départ et l'arrivée. Appelle : dist_lat_lon_deg