Relazione del progetto **Roundabout**

Lorenzo Rizzi, Simone Naglieri, Veronica Mungai

A.A. 2021-2022

1 Roundabout: cos'è

Il nostro progetto si ripropone di simulare in maniera semplice il traffico veicolare che si crea all'interno di una rotonda e nelle strade che vi confluiscono. Compilando le due translation unit (*main.cpp* e *functionsclasses.cpp*) insieme (aggiungendo *-lsfml-graphics -lsfml-window -lsfml-system*) ed eseguendo il file, il programma stamperà a schermo l'immagine di una rotonda in 2D vista dall'alto e un numero di strade equidistanziate definibili dall'utente. Dalle strade verranno generate in maniera casuale delle macchine (pallini blu) che si dirigeranno verso la rotonda rispettando una distanza minima di sicurezza. Quando una macchina sta per immettersi, controlla che in rotonda, entro un certo angolo modificabile, non ci siano altre macchine nelle vicinanze (davanti e dietro) e, nel caso, si arresta per dare la precedenza. Ogni macchina sa in anticipo a quale uscita abbandonare la rotonda e, quando vi arriva, si immette nella strada di uscita.

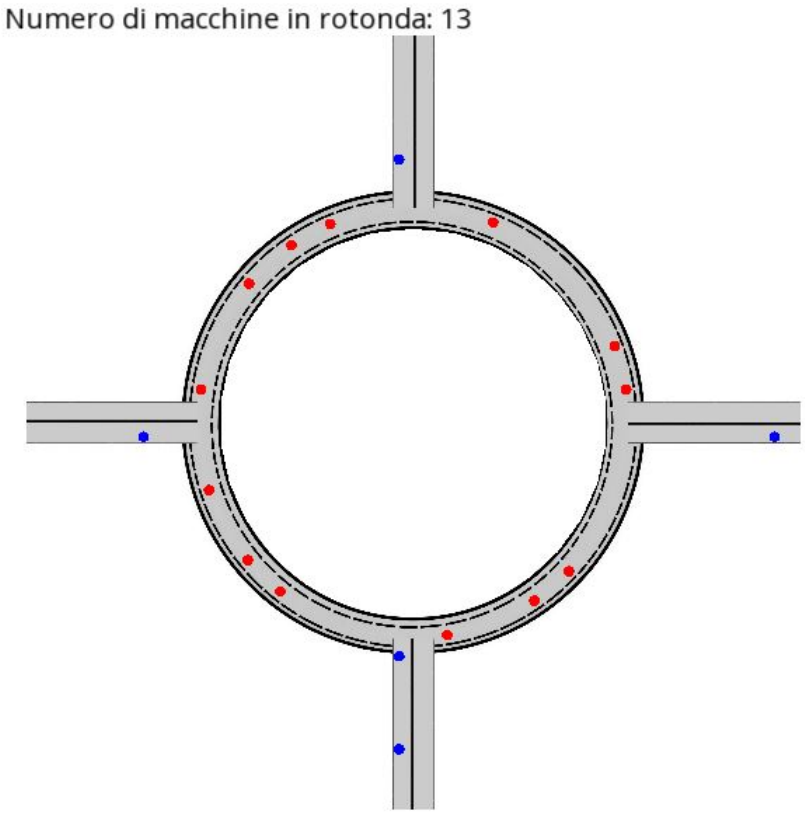

Figura 1: Una volta eseguito, il programma dovrebbe aprire una finestra grafica e far cominciare in automatico la simulazione del traffico.

2 Classi principali

Le classi principali implementate sono quelle di *car* (macchina), *rbout* abbreviazione per *roundabout* (rotonda) e *road* (strada), implementate in questo ordine affinché le funzioni membro di *road* possano utilizzare anche le variabili di *car* e *rbout* mentre quelle di *rbout* anche quelle di *car*.

2.1 Car

Car è la classe da cui dipende l'intero progetto e le sue variabili interne sono:

- **theta_**: (di tipo double) rappresenta l'angolo risultante dalla posizione della macchina nella rotonda secondo la schematizzazione in figura

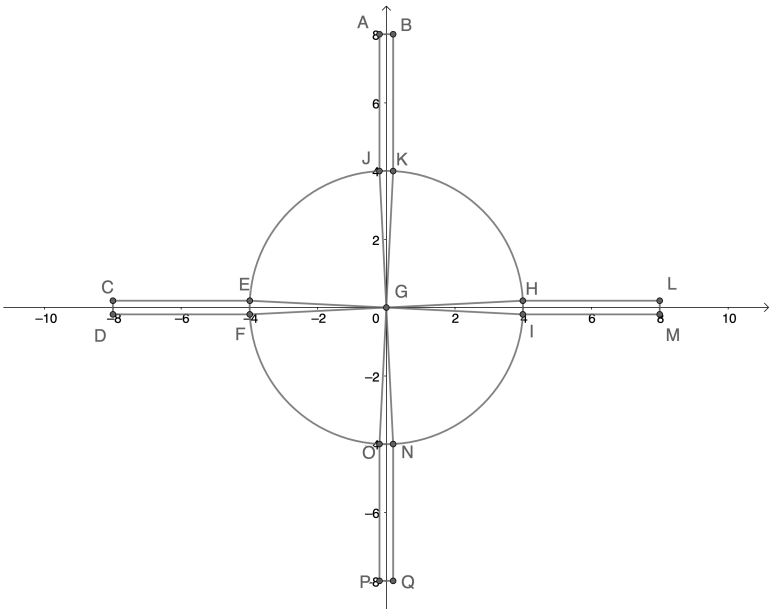

Figura 2: *Ad esempio, ipotizzando che questo schema sia una buona approssimazione della rotonda e delle strade che vi si immettono (ne prendiamo quattro per semplicità), una macchina posizionata nel punto H avrà un valore di theta_ un poco maggiore di 0, mentre una posizionata nel punto I sarà ad un angolo un poco inferiore di* 2π*. Tutti gli altri angoli sono calcolati di conseguenza.*

  Dal momento che in una stessa strada dovevano esserci sia le macchine in uscita che quelle in entrata, per evitare sovrapposizioni e/o malfunzionamenti nel caso di car entranti e uscenti dalla stessa road, è stato introdotto un offset di 0.066 radianti tra il centro della strada e ciascuna corsia. Nello specifico, l'angolo in figura *IGH* vale 2 volte l'offset.

  Si è inoltre scelto di esprimere *theta_* in radianti data la comodità di avere sempre a disposizione nella Standard Library il valore del π; tuttavia per quanto riguarda SFML è stata necessaria la conversione in gradi per l'orientamento delle strade nella finestra.

- **t_**: (di tipo double) rappresenta il grado di avanzamento della macchina lungo la strada. $t\_$ può quindi variare da 0 a 1 dove a $t\_ = 0$ la macchina si trova idealmente all'inizio della strada, mentre a $t\_ = 1$ la macchina è posizionata all'imboccatura della rotonda.

- **exit_**: (di tipo intero) rappresenta l'uscita a cui deve uscire la car. Il valore di *exit_* viene assegnato all'ingresso della macchina nella rotonda attraverso una specifica funzione [v. par. 3.2 *Funzioni libere*]. Varia tra 1 (che corrisponde alla strada all'angolo 0) e il numero delle strade [vedi variabili di rbout al par. 2.2].

- **can_I_enter_**: (di tipo bool) necessaria per il corretto ingresso delle macchine in rotonda per evitare sovrapposizioni con auto sulla strada e/o in rotonda. Per maggiori dettagli sul suo funzionamento vedere la funzione **can_I_enter_Y** [par. 3.1.1 *Funzioni membro, Car*].

2.2 Rbout

Questa classe serve per istanziare la rotonda, che rimane la stessa per tutta l'esecuzione del programma. È caratterizzata dalle seguenti variabili interne:

- **n_roads_**: (di tipo intero) rappresenta il numero delle strade da cui le macchine possono accedere alla rotonda. Per ovvi motivi pratici *n_roads* non può essere inferiore a 1.

- **radius_**: (di tipo double) è il raggio della rotonda.

Inoltre è stata necessaria l'introduzione di un vettore di car chiamato **car_rbout** che di ciclo in ciclo aggiornasse il numero di macchine in rotonda in base a quelle entranti e in uscita.

2.3 Road

Per ultima la classe rappresentante la strada; segue l'unica variabile interna da cui è caratterizzata assieme ai due vettori di car necessari per un corretto e più efficiente funzionamento del programma:

- **angle_**: (di tipo double) l'angolo della strada (in radianti), ipotizzando che tutte le strade siano alla stessa distanza l'un l'altra, calcolato allo stesso modo in cui viene determinato *theta_* per gli oggetti car.
- **car_in**: vettore di car relativo alle macchine in strada che entreranno nella rotonda.
- **car_out**: vettore di car per le auto nella strada uscite dalla rotonda.

3 Funzioni principali

3.1 Funzioni membro delle classi

3.1.1 Car

- Le funzioni **car::theta()**, **car::t()**, **car::exit()**, **car::can_I_enter()** permettono l'accesso ai membri privati dell'oggetto car.
- void **can_I_enter_Y** ()

  È la funzione che si occupa di far avanzare le macchine nell'ultimo pezzo della strada quando hanno ottenuto il permesso di immettersi in rotonda. A livello di codice, la funzione setta il parametro privato *can_I_enter_* da false a true.
- void **evolve_tplus** ()

  Incrementa il valore di $t\_$ di un dt.
- void **evolve_tminus** ()

  Decrementa il valore di $t\_$ di un dt.
- void **evolve_ang** ()

  Incrementa il valore di *theta_* di un $d\theta$.

3.1.2 Rbout

- Le funzioni **rbout::n_roads()** e **rbout::rad()** permettono di accedere al di fuori dello scope della classe ai membri privati (rispettivamente *n_roads_* e *radius_*).
- La funzione **rbout::carrbout()** permette di accedere al vettore membro privato *car_rbout*.
- La funzione **rbout::size_rbout** restituisce il numero di macchine presenti nella rotonda mentre la funzione **rbout::empty_rbout** valuta se nella rotonda non è presente alcuna macchina.
- void **newcar_rbt** (double const *street_angle*, int const *mean*)

  Quando chiamata su di un oggetto rbout, la funzione **newcar_rbt** crea un oggetto car e lo inizializza all'ingresso della strada situata nell'angolo *street_angle* settando come parametro *exit_* un valore casuale tramite chiamata alla funzione **random_call** [v. par. 3.2].
- void **evolve_rbt** ()

  Consente l'evoluzione delle macchine dentro il vettore *car_rbout* facendo una chiamata alla funzione membro di car **evolve_ang** per ogni macchina in rotonda che non si trovi già all'uscita corretta.

- int **transfer_rbt** ()

  È la funzione che si occupa di gestire il trasferimento dalla rotonda all'uscita della strada. Per ogni macchina presente nel vettore *car_rbout*, valuta se la condizioni di arrivo all'uscita corretta è verificata e, nel caso, ritorna come output il numero dell'uscita (il parametro *exit_*). Se nessuna macchina deve uscire, ritorna 0.

- void **erase_rbt** ()

  In maniera analoga a **transfer_rbt**, valuta per ogni macchina in rotonda se è stato raggiunto l'angolo di uscita e, nel caso, la rimuove dal vettore *car_rbout*.

3.1.3 Road

- Le funzioni **road::angle()**, **road::carin()**, **road::carout()** permettono di accedere ai membri privati dell'oggetto road.

- Le funzioni **road::size_in** e **road::size_out** ritornano il numero di macchine rispettivamente in *car_in* e in *car_out*.

- Le funzioni **road::empty_in**() e **empty_out**() valutano se i vettori car_in e car_out sono vuoti e, nel caso, restituiscono true.

- void **newcar_rd** (bool const *input*, int *rate*)

  La funzione assolve a due compiti. Se il parametro *input* è true, si occupa della generazione di nuove macchine in *car_in*; in particolare, se nella strada non ci sono più di 9 macchine e se la funzione **can_generate**(*rate*) restituisce true, crea un oggetto car situato a $t\_ = 0$ e lo inserisce in *car_in*. Al contrario, se *input* == false, la funzione si occupa di generare macchine in uscita (dunque nel vettore *car_out*) settando correttamento i parametri (ad esempio l'angolo, pari all'angolo della strada di competenza meno l'offset di ≈ 0.066 radianti).

- void **evolve_rd** (bool const *input*, rbout& *roundabout*, double const *min_ang*)

  Come sopra, il parametro input permette di distinguere i casi di macchine in *car_in* e in *car_out*. Nel primo caso, fa avanzare tutte le macchine nel vettore chiamando la funzione **car::evolve_tplus** se non devono ancora entrare in rotonda e se la distanza fra due macchine è maggiore di un valore di soglia. Analogamente, nel secondo caso la funzione fa avanzare le macchine in *car_out* facendo una chiamata a **car::evolve_tminus**.

- bool **transfer_rd**()

  Questa funzione si occupa di valutare se qualche macchina in *car_in* sta per entrare in rotonda (cioè $t\_ > 1$). Nel caso, restituisce true e la elimina dal vettore di appartenenza.

- void **erase_rd**()

  Questa funzione si occupa di eliminare le macchine definitivamente uscite dalla strada. Se il parametro di una macchina $t\_$ scende sotto lo zero, la toglie dal vettore *car_out*.

3.2 Funzioni libere

- bool **can_generate** (int const *rate*)

  È la funzione chiamata da **road::newcar_rd** per l'istanziamento di oggetti car. Genera un numero in base a una distibuzione uniforme [1] tra 1 e 1000 e nel caso questo numero sia minore o uguale del parametro *rate* passato alla funzione ritorna true, altrimenti false.

- int **random_call** (int const *max_exit*)

  Viene chiamata da **rbout::newcar_rbt** nell'istanziamento di un oggetto car per inizializzare il valore del parametro *exit_*. Genera un numero in base a una distribuzione poissoniana tra 1 e *max_exit* con valore di media *mean_exit* e ritorna l'intero ottenuto.[2]

[1] Di default, il programma genera secondo una distribuzione uniforme. È comunque possibile intervenire sul codice scegliendo una distribuzione a proprio piacimento.

[2] Come sopra, è possibile modificare la distribuzione delle uscite.

- double **module360** (double *angle*)

  Fa sì che tutti gli angoli superiori a 2π vengano riportati in un intervallo tra 0 (compreso) e 2π (non compreso). Prende come parametro l'angolo in questione e ne ritorna il valore una volta modificato.

- bool **is_free** (double const *street_angle*, rbout& *roundabout*, double const *min_ang*)

  È la funzione che si occupa del corretto ingresso delle macchine all'interno della rotonda. Prima dell'ingresso di un oggetto car viene controllato in tutto il vettore *car_rbout* che non ci siano macchine tali per cui la differenza delle rispettive variabili *theta_* sia minore del parametro *min_ang*. Per la strada a 0 radianti, affinché venga rispettata questa distanza minima è stata aggiunta la condizione per cui se la differenza dei *theta_* supera i 360°-*min_ang* l'auto non può entrare in rotonda. Nel caso in cui l'oggetto car non possa entrare viene ritornato false, altrimenti true.

4 Funzionamento generale

Nel file *main.cpp* è possibile accedere a tutti i parametri del progetto e modificarli per ottenere l'esecuzione del programma con dinamiche leggermente diverse. Di seguito i parametri del progetto:

- **rate**: parametro della funzione **can_generate** per l'istanziamento o meno di oggetti car.
- **minimum_angle**: angolo rappresentante la distanza minima tra le auto in rotonda.
- **fps**: frame al secondo.
- **N_ROADS**: numero delle strade della rotonda.
- **mean_exit**: numero della strada con più alta probabilità di uscita.

Dopo che sono stati inizializzati i parametri del programma vengono istanziati una rotonda e un vettore di *N_ROADS* strade ciascuna con i rispettivi membri interni. Una volta creata una finestra con la rotonda e le relative strade, il font e le texture, in un ciclo-for a ciascuna di esse vengono applicati i metodi **road::newcar_rd**(*true*, *rate*) (per la generazione di macchine nelle strade) e **road::evolve_rd**(*true*, *roundabout*, *minimum_angle*) (per permettere ad esse di muoversi lungo le road). Se **road::tranfer_rd** restituisce true, il metodo **rbout::newcar_rbt** viene appicato alla rotonda istanziata in precedenza.

In rotonda ciascuna car avanza grazie al metodo **rbout::evolve_rbt**, chiamato sulla rotonda stessa. L'uscita avviene se la funzione **rbout::tranfer_rbt** restituisce un valore maggiore di 0, a questo punto viene chiamato il metodo **road::newcar_rd**(*false*, *rate*) sulla strada corretta e, una volta che la car è giunta alla fine della strada, viene applicato il metodo **rbout::erase_rbt** per la sua cancellazione.

A ogni ciclo viene inoltre stampato il numero di macchine in rotonda per facilitare la comprensione dell'utente durante l'esecuzione del programma.

5 Test

I test realizzati tramite DocTest (compilare assieme *functionsclasses.cpp* e *functionsclasses_test.cpp*) ci hanno permesso di valutare l'effettivo funzionamento del programma per come lo avevamo progettato e la generale mancanza di errori a livello logico. Alcuni test miravano a valutare la correttezza di singole funzioni (ad esempio il testing delle funzioni **size_in**() e **size_out**()) mentre altri test più complessi sono stati implementati per confermare la giusta evoluzione delle macchine all'interno del programma. A titolo d'esempio, esiste un test il cui compito è quello di controllare se, generata volutamente una macchina in rotonda con uscita prefissata, questa rispetta la condizione corretta di uscita dalla rotonda e non supera la strada a lei destinata. Tutti i test proposti sono stati superati con successo.

6 Annotazioni finali

Il programma in fase di compilazione (usando *-Wall -Wextra -fsanitize=address*) non presenta né errori di compilazione né warning di alcun genere fatta eccezione per questo:

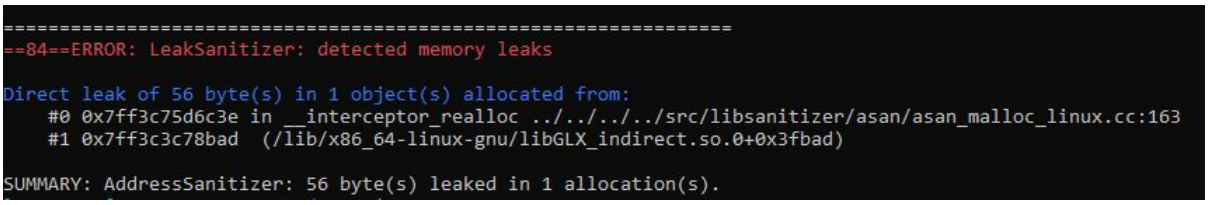

Figura 3: *Errore di memory leak come segnalato sul terminale dopo la chiusura della finestra grafica in seguito all'esecuzione del programma con ./nome_eseguibile.*

Dal momento che, però, questo memory leak veniva rilevato solamente dai computer di alcuni membri del gruppo (due su tre) si è indagato meglio sulla sua provenienza e, visto che non sono stati utilizzati raw arrays o puntatori gestiti direttamente dal programmatore, né si è utilizzata, al di là dei vettori, l'allocazione dinamica, a seguito di ricerche e approfondimenti è risultato che questo genere di errori può essere imputato con buona probabilità ai drivers grafici di sistema (ed infatti il messaggio di errore varia da sistema a sistema).