QPathfinder Traffic Simulator

Examples of using tool are on Youtube:
https://www.youtube.com/watch?v=yQyGyulcV-A
https://www.youtube.com/watch?v=T3PtZV-afxk

AUTHOR asdasd97

Version 1.0
Sun Sep 27 2020

Table of Contents

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

Data Structure Documentation

GraphData Class Reference

A collection of nodes, paths and objects link to them

Data Structures

- struct **SavingStructure**
  Structure used to save data between play and edit mode

Public Member Functions

- void **SaveTimers** ()
  Method gather data to structure and save it to file

- void **LoadTimers** ()
  Method load file and fill certain objects

- **Path GetPathBetween** (**Node** from, **Node** to)
  Method returns path that connect node "from "to"

- void **Clear** ()
  Metod delete all junctions and streets

- void **ReGenerateIDs** ()
  The method is responsible for assigning identifiers to nodes and paths and creating a dictionary mapping a pair of node IDs, ID of path

Data Fields

- List< **Street** > **allStreets** = new List<**Street**>()
  List of all streets

- List< **Junction** > **allJunctions** = new List<**Junction**>()
  List of all junctions

- List< **Node** > **nodes** = new List<**Node**>()
  List of all nodes

- List< **Node** > **centers** = new List<**Node**>()
  List of center nodes

- List< **Path** > **paths** = new List<**Path**>()
  List of all paths

- Dictionary< Vector2Int, int > **pathsByNodes** = new Dictionary<Vector2Int, int>()
  Dictionary of path ID by pair of Nodes ID

Detailed Description

A collection of nodes, paths and objects link to them

QPathFinder modified

Member Function Documentation

void Clear ()

Metod delete all junctions and streets

Path GetPathBetween (Node *from*, Node *to*)

Method returns path that connect node "from "to"

Parameters

from	node "from"
to	node "to"

Returns

Path that connect nodes

void LoadTimers ()

Method load file and fill certain objects

void ReGenerateIDs ()

The method is responsible for assigning identifiers to nodes and paths and creating a dictionary mapping a pair of node IDs, ID of path

QPathFinder modified

void SaveTimers ()

Method gather data to structure and save it to file

Field Documentation

List<Junction> allJunctions = new List<Junction>()

List of all junctions

List<Street> allStreets = new List<Street>()

List of all streets

List<Node> centers = new List<Node>()

List of center nodes

List<Node> nodes = new List<Node>()

List of all nodes

List<Path> paths = new List<Path>()

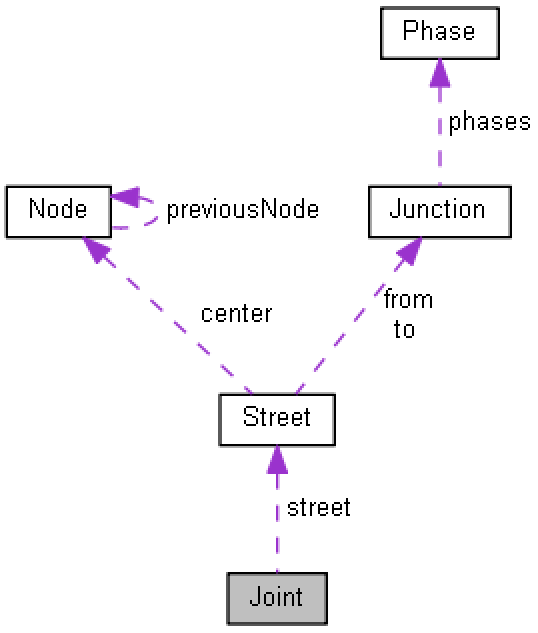
List of all paths

Dictionary<Vector2Int, int> pathsByNodes = new Dictionary<Vector2Int, int>()

Dictionary of path ID by pair of Nodes ID

Joint Class Reference

Part of the street connected to the intersection
Collaboration diagram for Joint:



Public Member Functions

- **Joint** (**Street Street**, bool Type)
  Construktor

Data Fields

- **Street street**
  Stores references to the street of which it is part
- List< **Node** > **input** = new List<**Node**>()
  Stores a list of nodes of paths entering an intersection
- List< **Node** > **output** = new List<**Node**>()
  Stores a list of vertices of paths exiting from an junction
- Vector3 **outsideVec**
  Stores the normalized vector from the center to the path junction

Properties

- Vector3 **Position** `[get]`
  Returns the approximate position of the joint

Detailed Description

Part of the street connected to the intersection

Constructor & Destructor Documentation

Joint (Street *Street*, bool *Type*)

Construktor

Parameters

Street	**Street** to which it belongs
Type	Connection type. True if junction "to", False if junction "from"

Field Documentation

List<Node> input = new List<Node>()

Stores a list of nodes of paths entering an intersection

List<Node> output = new List<Node>()

Stores a list of vertices of paths exiting from an junction

Vector3 outsideVec

Stores the normalized vector from the center to the path junction

Street street

Stores references to the street of which it is part

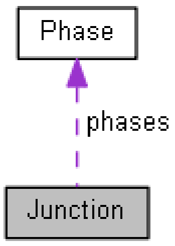
Property Documentation

Vector3 Position `[get]`

Returns the approximate position of the joint

Junction Class Reference

Junction
Inherits MonoBehaviour.
Collaboration diagram for Junction:



Public Member Functions

- void **AddJoint** (**Joint** joint)
  The method is responsible for adding a new joint

- void **RemoveJoint** (**Street** street)
  The method is responsible for removing a new joint

- void **Calculate** ()
  The method creates paths between joints and adds them to the junction phases

- void **Destroy** ()
  The method is responsible for the destruction of object and attached streets

- void **Select** (bool light=true)
  The method is responsible for highlighting the junction as marked

Data Fields

- List< **Joint** > **joints** = new List<**Joint**>()
  List of joints

- List< **Path** > **paths** = new List<**Path**>()
  Store paths created by junction

- **Phase**[] **phases**
  Stores an array of phases of this junction

- float **timeToPhase** = 0f
  Stores the time until the junction phase changes

- float **margin** = 0
  Stores margin of distance between joint and junction

- float **cycleTime** = 20f
  Store time of full cycle of junction phases

- float[] **timers**
  Stores an array of duration of each phase

- bool **timersCalc** = true
  Is junction should calculate duration of phases

Properties

- bool **Rondo** `[get, set]`
  Mark is this junction is roundabout

Private Member Functions

- void **Start** ()
  The method that is run when the simulation starts. Decides whether the intersection will act as a traffic light or a roundabout.

- IEnumerator **PhaseChanger** ()
  Routine that is responsible for the operation of the light intersection

- IEnumerator **RondoRutine** ()
  Routine that is responsible for the operation of the roundabout

- bool **IsFree** ()
  The method determines whether the intersection is abandoned

- List< **Path** >[] **StreetWays** (int curentIndex, bool individual=false)
  Creates paths from one joint to others

- void **SortStreets** ()
  The method is responsible for sorting the join list, counterclockwise to the junction

- void **Clear** ()
  The method is responsible for removing owned paths

Private Attributes

- int **phase** = 0
  Number of current running phase

Detailed Description

Junction

Object connecting streets with a traffic light or roundabout

Member Function Documentation

void AddJoint (Joint *joint*)

The method is responsible for adding a new joint

Parameters

joint	**Joint** of street

void Calculate ()

The method creates paths between joints and adds them to the junction phases

void Clear () `[private]`

The method is responsible for removing owned paths

void Destroy ()

The method is responsible for the destruction of object and attached streets

bool IsFree () `[private]`

The method determines whether the intersection is abandoned

Returns

True if all paths all ababdoned

IEnumerator PhaseChanger () `[private]`

Routine that is responsible for the operation of the light intersection

void RemoveJoint (Street *street*)

The method is responsible for removing a new joint

Parameters

street	**Street** with joint is part of

IEnumerator RondoRutine () `[private]`

Routine that is responsible for the operation of the roundabout

void Select (bool *light* = `true`)

The method is responsible for highlighting the junction as marked

Parameters

light	true if highlighting

void SortStreets () `[private]`

The method is responsible for sorting the join list, counterclockwise to the junction

void Start () `[private]`

The method that is run when the simulation starts. Decides whether the intersection will act as a traffic light or a roundabout.

List<Path> [] StreetWays (int *curentIndex*, bool *individual* = `false`) `[private]`

Creates paths from one joint to others

Parameters

curentIndex	Index of the street from which the vehicles are entering
individual	Determines whether the street will be in the individual phase

Returns

A three-element array of path lists coming out of the joint. These are lists of paths to the left, front, and right

---

Field Documentation

float cycleTime = 20f

Store time of full cycle of junction phases

List<Joint> joints = new List<Joint>()

List of joints

float margin = 0

Stores margin of distance between joint and junction

List<Path> paths = new List<Path>()

Store paths created by junction

int phase = 0`[private]`

Number of current running phase

Phase [] phases

Stores an array of phases of this junction

float [] timers

Stores an array of duration of each phase

bool timersCalc = true

Is junction should calculate duration of phases

float timeToPhase = 0f

Stores the time until the junction phase changes

Property Documentation

bool Rondo`[get]`, `[set]`

Mark is this junction is roundabout

Node Class Reference

Single **Node**. From which will be created Paths

Collaboration diagram for Node:

Public Member Functions

- **Node** (Vector3 Position)
  Construktor

Data Fields

- Vector3 **position**
  Position of node
- int **ID** = -1
  ID of node
- float **heuristicDistance**
  Distance calculated by heuristic
- float **pathDistance**
  Distance from first node
- **Node previousNode**
  Distance from previous node

Properties

- float **CombinedHeuristic** `[get]`
  Return sum of distance

Detailed Description

Single **Node**. From which will be created Paths

QPathFinder

Constructor & Destructor Documentation

Node (Vector3 *Position*)

Construktor

Parameters

Position	Position of node

Field Documentation

float heuristicDistance

Distance calculated by heuristic

QPathFinder

int ID = -1

ID of node

float pathDistance

Distance from first node

QPathFinder

Vector3 position

Position of node

Node previousNode

Distance from previous node

QPathFinder

Property Documentation

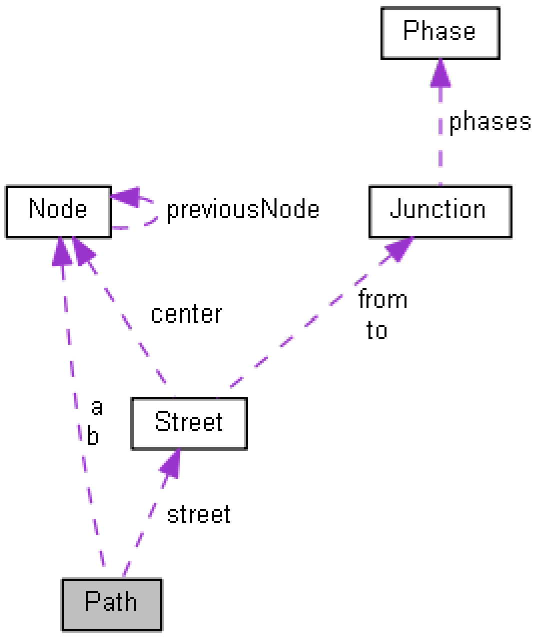
float CombinedHeuristic `[get]`

Return sum of distance

QPathFinder

Path Class Reference

Path is a connection between 2 Nodes.
Collaboration diagram for Path:



Public Member Functions

- **Path** (**Node** A, **Node** B, Transform Parent, HidePath Hide=HidePath.Shown, BlockType Prioritet=BlockType.Open)
  Construktor

- void **Visualize** ()
  The method is responsible for the visual representation of the object

- bool **CanEnter** (BlockType Priority)
  Answers that vehicle can or can not enter to this path

- int **EnterQueue** ()
  The method is responsible for adding vehicle to queue

- void **LeaveQueue** ()
  The method is responsible for delete vehicle to queue

Data Fields

- int **leftQueue** = 0
  Counts vehicles that leave this path

- int **entireQueue** = 0
  Counts vehicles that enter this path

- int **maxInQueue**
  Stores maximum number of vehicles that can stay on path in same time

- int **autoGeneratedID**
  Store ID

- BlockType **priority**
  Marks type of path priority

- BlockType **block** = BlockType.Open
  Marks path as open, open for prioritized, blocked

- HidePath **hide**
  Marks type of hiden path

- **Street street**
  Store references to the street, if it belongs to any

- Transform **transform**
  Stores references to a Transform component, if it has one

Properties

- int? **IDOfA** `[get]`
  Returns the identifier of the first node

- int? **IDOfB** `[get]`
  Zwraca identyfikator drugiego wierzchoÅ‚ka

- Vector3? **PosOfA** `[get]`
  Returns the identifier of the second node

- Vector3? **PosOfB** `[get]`
  Returns the position of the second node

- int **CurrentQueue** `[get]`
  Returns current amount of vehicles i queue

- float **Cost** `[get]`
  Returns current cost of travel this ptah

- float **SumaryWaitingTime** `[get]`
  Returns sum of times that vehicles wait in queue

Private Attributes

- **Node a**
  Stores first node of path

- **Node b**
  Stores second node of path

- float **cost**
  Stores cost of traveling this path

- List< float > **queueTimes**
  Keeps a list of the times the vehicle entered the track

Detailed Description

Path is a connection between 2 Nodes.

QPathFinder modified

Constructor & Destructor Documentation

Path (Node *A*, Node *B*, Transform *Parent*, HidePath *Hide* = `HidePath.Shown`, BlockType *Prioritet* = `BlockType.Open`)

Construktor

Parameters

A	First node
B	Second node
Parent	Paretn of object
Hide	Hide mark
Prioritet	Priority mark

Member Function Documentation

bool CanEnter (BlockType *Priority*)

Answers that vehicle can or can not enter to this path

Parameters

Priority	Priority of path of asking vehicle

Returns

Returns true if vehicle can enter

int EnterQueue ()

The method is responsible for adding vehicle to queue

Returns

Returns number of vehicles in queue

void LeaveQueue ()

The method is responsible for delete vehicle to queue

void Visualize ()

The method is responsible for the visual representation of the object

---

Field Documentation

Node a`[private]`

Stores first node of path

int autoGeneratedID

Store ID

Node b`[private]`

Stores second node of path

BlockType block = BlockType.Open

Marks path as open, open for prioritized, blocked

float cost`[private]`

Stores cost of traveling this path

int entireQueue = 0

Counts vehicles that enter this path

HidePath hide

Marks type of hiden path

int leftQueue = 0

Counts vehicles that leave this path

int maxInQueue

Stores maximum number of vehicles that can stay on path in same time

BlockType priority

Marks type of path priority

List<float> queueTimes `[private]`

Keeps a list of the times the vehicle entered the track

Street street

Store references to the street, if it belongs to any

Transform transform

Stores references to a Transform component, if it has one

Property Documentation

float Cost `[get]`

Returns current cost of travel this ptah

int CurrentQueue `[get]`

Returns current amount of vehicles i queue

int? IDOfA `[get]`

Returns the identifier of the first node

int? IDOfB `[get]`

Zwraca identyfikator drugiego wierzchoÅ‚ka

Vector3? PosOfA `[get]`

Returns the identifier of the second node

Vector3? PosOfB `[get]`

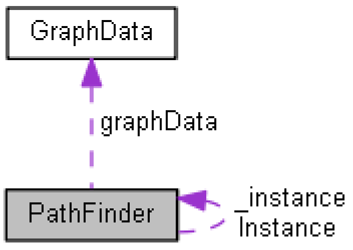
Returns the position of the second node

float SumaryWaitingTime `[get]`

Returns sum of times that vehicles wait in queue

PathFinder Class Reference

The class is responsible for creating vehicles and finding the shortest route
Inherits MonoBehaviour.
Collaboration diagram for PathFinder:



Data Fields

- int **amount** = 0
  Stores the current number of vehicles

- int **maxCars** = 100
  Stores the maximum number of vehicles

- float **spawnFrequency** = 0.1f
  Stores how often vehicles are created

- float **workDelay** = 5f
  Stores the amount of time vehicles spend at the workplace

- float **shopingDelay** = 1f
  Stores the amount of time vehicles spend in a trading place

- bool **randomSpawn** = false
  Determines whether vehicles will be created from random streets for random purposes or according to a schedule

- bool **spawning** = true
  Specifies whether the object is to create vehicles

- bool **save** = true
  Determines whether the object should save data at the end of the simulation

- bool **calculateTimers** = true
  Specifies whether intersections should calculate the duration of each phase

- bool **drawPaths** = true
  Specifies whether to display path lines

- bool **showSpawns** = true

Specifies whether to show information about vehicle creation points

- **GraphData graphData** = new **GraphData**()
  Stores the graph data

Properties

- int **TimeScale** `[get, set]`
  Defines the pace of the simulation

Private Member Functions

- void **Awake** ()
  The method, which is run before the simulation starts, prepares the data for simulation

- void **OnDestroy** ()
  Method run when the object is destroyed

- List< int >[] **MakeSpawnList** ()
  The method creates an array of places from which vehicles will leave and arrive

- void **SpawnPredictably** (List< int >[] spawns)
  The method is responsible for creating the vehicle according to the schedule

- void **SpawnRandom** ()
  The method is responsible for creating a vehicle going to a random destination

- **PathFollower SpawnCar** ()
  The method is responsible for creating the vehicle object

- List< **Path** > **RandomPath** ()
  The method is responsible for finding a route to a random node

- IEnumerator **Spawn** ()
  Routine is responsible for creating vehicles

- IEnumerator **RemoveCars** ()
  Routine is responsible for removing vehicles that have reached their destination

- List< **Path** > **NodesToPath** (List< **Node** > nodes)
  The method is responsible for converting the list of vertices into a list of paths

- List< **Node** > **FindShortedPathSynchronousInternal** (int fromNodeID, int toNodeID)
  The method is responsible for finding the path between two vertices

- IEnumerator **FindShortestPathAsynchonousInternal** (int fromNodeID, int toNodeID, System.Action< List< **Node** >> callback)

Private Attributes

- List< Transform > **cars** = new List<Transform>()
  Stores a list of created vehicles

Detailed Description

The class is responsible for creating vehicles and finding the shortest route

QPathFinder modified

Member Function Documentation

void Awake () `[private]`

The method, which is run before the simulation starts, prepares the data for simulation

QPathFinder

List<Node> FindShortedPathSynchronousInternal (int *fromNodeID*, int *toNodeID*) `[private]`

The method is responsible for finding the path between two vertices

IEnumerator FindShortestPathAsynchonousInternal (int *fromNodeID*, int *toNodeID*, System.Action< List< Node >> *callback*) `[private]`

QPathFinder not used

Parameters

fromNodeID	
toNodeID	
callback	

List<int> [] MakeSpawnList () `[private]`

The method creates an array of places from which vehicles will leave and arrive

Returns

A array storing 5 lists of places of creation: visitors, residents, commercial places, workplaces, people leaving

List<Path> NodesToPath (List< Node > *nodes*) `[private]`

The method is responsible for converting the list of vertices into a list of paths

Parameters

nodes	List of nodes

Returns

List of paths

void OnDestroy () `[private]`

Method run when the object is destroyed

QPathFinder

List<Path> RandomPath () `[private]`

The method is responsible for finding a route to a random node

Returns

List of paths

IEnumerator RemoveCars () `[private]`

Routine is responsible for removing vehicles that have reached their destination

IEnumerator Spawn () `[private]`

Routine is responsible for creating vehicles

Returns

PathFollower SpawnCar () `[private]`

The method is responsible for creating the vehicle object

Returns

Vehicle

void SpawnPredictably (List< int >[] *spawns*) `[private]`

The method is responsible for creating the vehicle according to the schedule

Parameters

spawns	List of vehicle creation locations

void SpawnRandom () `[private]`

The method is responsible for creating a vehicle going to a random destination

Field Documentation

int amount = 0

Stores the current number of vehicles

bool calculateTimers = true

Specifies whether intersections should calculate the duration of each phase

List<Transform> cars = new List<Transform>() `[private]`

Stores a list of created vehicles

bool drawPaths = true

Specifies whether to display path lines

GraphData graphData = new GraphData()

Stores the graph data

int maxCars = 100

Stores the maximum number of vehicles

bool randomSpawn = false

Determines whether vehicles will be created from random streets for random purposes or according to a schedule

bool save = true

Determines whether the object should save data at the end of the simulation

float shopingDelay = 1f

Stores the amount of time vehicles spend in a trading place

bool showSpawns = true

Specifies whether to show information about vehicle creation points

float spawnFrequency = 0.1f

Stores how often vehicles are created

bool spawning = true

Specifies whether the object is to create vehicles

float workDelay = 5f

Stores the amount of time vehicles spend at the workplace

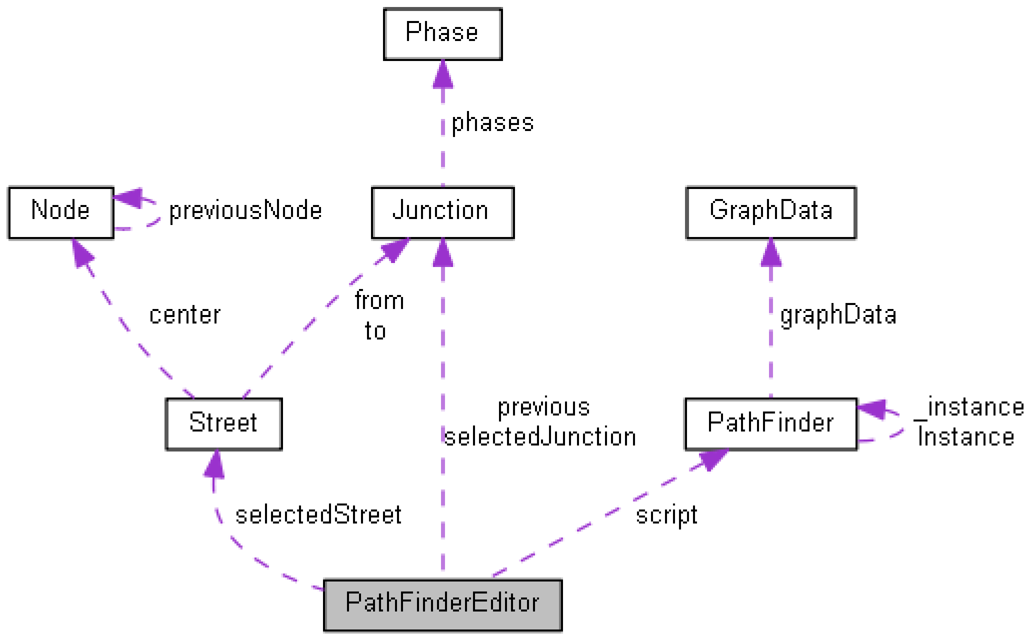
Property Documentation

int TimeScale`[get]`**,** `[set]`

Defines the pace of the simulation

PathFinderEditor Class Reference

Class is responsible for displaying GUI
Inherits Editor.
Collaboration diagram for PathFinderEditor:



Private Member Functions

- void **OnSceneGUI** ()
  Method running when user using scene window

- void **DrawGUIWindowOnScene** ()
  Method is responsible for display window in scene window

- void **DrawPathLine** ()
  The method is responsible for the visualization of paths

- void **UpdateMouseInput** ()
  The method is responsible for capturing mouse events

- void **OnMouseClick** (Vector2 mousePos)
  The method is responsible for the reaction to the click of the mouse

- **Junction CreateJunction** (Vector3 position)
  Method is responsible for creating junction

- void **CreateStreet** (**Junction** a, **Junction** b)
  Method responsible for creating new street

- void **DeleteJunction** (**Junction** junction)

Method responsible for deleting junction

- void **DeleteStreet** (**Street** street)
  Method responsible for deleting street

- void **RefreshData** ()
  Method responsible for recalculate streets, junctions and refresh IDs of paths and nodes

- void **ClearAll** ()
  Deletes all graph data

- void **OnEnable** ()
  Method running when object is Enable

- void **OnDisable** ()
  Method running when object is Disable

- void **OnPlayModeStateChanged** (PlayModeStateChange state)
  Method runs when simulation starts or ends

Private Attributes

- SceneMode **sceneMode**
  Selected scene mode

Detailed Description

Class is responsible for displaying GUI

QPathFinder modified

Member Function Documentation

void ClearAll () `[private]`

Deletes all graph data

Junction CreateJunction (Vector3 *position*) `[private]`

Method is responsible for creating junction

Parameters

position	Position where junction should be created

void CreateStreet (Junction *a*, Junction *b*)`[private]`

Method responsible for creating new street

Parameters

a	First junction
b	Second junction

void DeleteJunction (Junction *junction*)`[private]`

Method responsible for deleting junction

Parameters

junction	Selected junction

void DeleteStreet (Street *street*)`[private]`

Method responsible for deleting street

Parameters

street	Selected **Street**

void DrawGUIWindowOnScene ()`[private]`

Method is responsible for display window in scene window

QPathFinder modified

void DrawPathLine ()`[private]`

The method is responsible for the visualization of paths

QPathFinder modified

void OnDisable ()`[private]`

Method running when object is Disable

void OnEnable ()`[private]`

Method running when object is Enable

QPathFinder modified

void OnMouseClick (Vector2 *mousePos*)`[private]`

The method is responsible for the reaction to the click of the mouse

QPathFinder modified

Parameters

mousePos	Position of cursor

void OnPlayModeStateChanged (PlayModeStateChange *state*)`[private]`

Method runs when simulation starts or ends

Parameters

state	State of simulation

void OnSceneGUI ()`[private]`

Method running when user using scene window

QPathFinder

void RefreshData ()`[private]`

Method responsible for recalculate streets, junctions and refresh IDs of paths and nodes

void UpdateMouseInput ()`[private]`

The method is responsible for capturing mouse events

QPathFindermodified

Field Documentation

SceneMode sceneMode`[private]`

Selected scene mode

PathFollower Class Reference

Vahicle

Inherits MonoBehaviour.

Public Member Functions

- void **Follow** (List< **Path** > **Path**, int ReturningType=-1, float WaitingTime=0, List< **Path** > ReturningPath=null)
  The method is responsible for assigning the travel route and its start

- void **StopFollowing** ()
  Responsible for abort the travel routine

Private Member Functions

- void **Awake** ()
  Method run when the object is created

- IEnumerator **FollowRoutine** (List< **Path** > path)
  Routine is responsible for driving the vehicle along the route

Private Attributes

- List< **Path** > **returningPath**
  Stores a list of paths for the vehicle to return to

Detailed Description

Vahicle

QPathFinder modified

Member Function Documentation

void Awake () `[private]`

Method run when the object is created

void Follow (List< Path > *Path*, int *ReturningType* = `-1`, float *WaitingTime* = `0`, List< Path > *ReturningPath* = `null`)

The method is responsible for assigning the travel route and its start

Parameters

Path	List paths of route
ReturningType	Type of return: -1 none, 2 from the place of sale, 3 from the place of work
WaitingTime	Time of stay at the destination
ReturningPath	List of paths return route

IEnumerator FollowRoutine (List< Path > *path*) `[private]`

Routine is responsible for driving the vehicle along the route

Parameters

path	List of paths making the route

void StopFollowing ()

Responsible for abort the travel routine

Field Documentation

List<Path> returningPath `[private]`

Stores a list of paths for the vehicle to return to

Phase Class Reference

Phase of junction

Public Member Functions

- **Phase** (Dictionary< int, string > dictionary, int mode, int i)
  Construktor

Data Fields

- List< int > **routes** = new List<int>()
  Store list of paths belonging to phase

- List< Vector2Int > **streetsPaths** = new List<Vector2Int>()
  Store list of pairs defining the streets and path from which traffic is coming

- float **queueTime** = 5f
  Store the time that vehicles wait for opening paths of this phase

---

Detailed Description

Phase of junction

It decides which paths are passable at any given time

Constructor & Destructor Documentation

Phase (Dictionary< int, string > *dictionary*, int *mode*, int *i*)

Construktor

Parameters

dictionary	Dictionary of track numbers and turn codes
mode	**Junction** modifier
i	**Phase** number that will be created

Field Documentation

float queueTime = 5f

Store the time that vehicles wait for opening paths of this phase

List<int> routes = new List<int>()

Store list of paths belonging to phase

List<Vector2Int> streetsPaths = new List<Vector2Int>()

Store list of pairs defining the streets and path from which traffic is coming

GraphData.SavingStructure Struct Reference

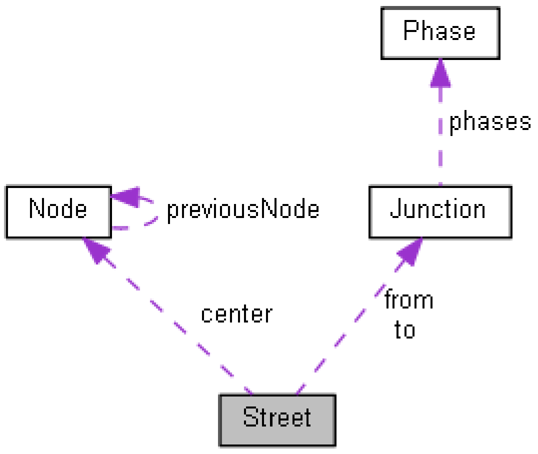
Structure used to save data between play and edit mode

Detailed Description

Structure used to save data between play and edit mode

Street Class Reference

Street
Inherits MonoBehaviour.
Collaboration diagram for Street:



Public Member Functions

- void **Init** (**Junction** From, **Junction** To, int fromCount=1, int toCount=1)
  The method is responsible for initiating the street

- void **Destroy** (**Junction** spare=null)
  The method is responsible for destroying the object

- void **Resize** ()
  The method is responsible for changing the length of the paths

- void **RecalcJunction** ()
  The method is responsible for calculating the connected junction

- void **Calculate** ()
  The method is responsible for creating a predetermined number of paths

- void **Select** (bool light=true)
  The method is responsible for highlighting the street as marked

Data Fields

- **Junction from**
  Store references to the "from" intersection

- **Junction to**
  Store references to the "to" intersection

- string **Name** = ""

Stores the street name

- List< **Path** > **paths** = new List<**Path**>()
  Store list of paths

- List< **Node** > **nodes** = new List<**Node**>()
  Store list of nodes

- **Node center**
  Holds the middle node

- int **iFrom** = 2
  Stores the number of paths "from-to"

- int **iTo** = 2
  Stores the number of paths "to-from"

- int[] **spawns** = new int[5] { 0, 0, 0, 0, 0 }
  It stores how many vehicles of a given type enter or leave this street It is an array that stores 5 types: visitors, residents, commercial places, workplaces, and people leaving

Private Member Functions

- void **Clear** ()
  The method is responsible for removing owned paths and node

- Vector3 **Perpendic** (**Junction** j)
  The method calculates the vector by which the paths leading to the junction will end

Private Attributes

- Vector3 **fromBorder**
  Stores position

Detailed Description

Street

Member Function Documentation

void Calculate ()

The method is responsible for creating a predetermined number of paths

void Clear () `[private]`

The method is responsible for removing owned paths and node

void Destroy (Junction *spare* = `null`)

The method is responsible for destroying the object

Parameters

spare	**Junction** that keep references to the street

void Init (Junction *From*, Junction *To*, int *fromCount* = `1`, int *toCount* = `1`)

The method is responsible for initiating the street

Parameters

From	**Junction** "from"
To	**Junction** "to"
fromCount	Number of paths "from-to"
toCount	Number of paths "to-from"

Vector3 Perpendic (Junction *j*) `[private]`

The method calculates the vector by which the paths leading to the junction will end

Parameters

j	The junction the vector will apply to

Returns

Scaled vector perpendicular to other junction streets

void RecalcJunction ()

The method is responsible for calculating the connected junction

void Resize ()

The method is responsible for changing the length of the paths

void Select (bool *light* = `true`)

The method is responsible for highlighting the street as marked

Parameters

light	True if highlight

Field Documentation

Node center

Holds the middle node

Junction from

Store references to the "from" intersection

Vector3 fromBorder `[private]`

Stores position

int iFrom = 2

Stores the number of paths "from-to"

int iTo = 2

Stores the number of paths "to-from"

string Name = ""

Stores the street name

List<Node> nodes = new List<Node>()

Store list of nodes

List<Path> paths = new List<Path>()

Store list of paths

int [] spawns = new int[5] { 0, 0, 0, 0, 0 }

It stores how many vehicles of a given type enter or leave this street It is an array that stores 5 types: visitors, residents, commercial places, workplaces, and people leaving

Junction to

Store references to the "to" intersection