Simulation

Generated by Doxygen 1.9.2

1	Namespace Index	1
	1.1 Packages	1
2	Hierarchical Index	3
	2.1 Class Hierarchy	3
3	Class Index	5
	3.1 Class List	5
4	File Index	7
	4.1 File List	7
5	Namespace Documentation	9
	5.1 Simulation Namespace Reference	9
6	Class Documentation	11
	6.1 Simulation.ClosestEncounterWatcher Class Reference	11
	6.1.1 Detailed Description	12
	6.1.2 Constructor & Destructor Documentation	12
	6.1.2.1 ClosestEncounterWatcher() [1/2]	12
	6.1.2.2 ClosestEncounterWatcher() [2/2]	12
	6.1.3 Member Function Documentation	13
	6.1.3.1 ResetClosestEncounter()	13
	6.1.3.2 Simulation_Stepped()	13
	6.1.4 Member Data Documentation	13
	6.1.4.1 DEFAULT_SETTINGS	13
	6.1.5 Property Documentation	14
	6.1.5.1 CurrentClosestEncounter	14
	6.2 Simulation.ClosestEncounterWatcherSettings Struct Reference	14
	6.2.1 Detailed Description	14
	6.2.2 Property Documentation	14
	6.2.2.1 AutoStopSimulation	14
	6.3 Simulation.ConstantStepSimulation Class Reference	15
	6.3.1 Detailed Description	15
	6.3.2 Constructor & Destructor Documentation	15
	6.3.2.1 ConstantStepSimulation()	15
	6.3.3 Member Function Documentation	16
	6.3.3.1 DoStep()	16
	6.4 Simulation.DynamicWorldPoint Class Reference	16
	6.4.1 Detailed Description	17
	6.4.2 Constructor & Destructor Documentation	17
		17
	6.4.2.1 DynamicWorldPoint() [1/2]	17
	6.4.3 Member Function Documentation	17
	0.4.5 Member Function Documentation	ıø

6.4.3.1 CloneToWorld()	18
6.4.3.2 StartApplyingForce()	18
6.4.3.3 Step()	19
6.4.3.4 StopApplyingForce()	19
6.4.4 Property Documentation	19
6.4.4.1 Acceleration	19
6.4.4.2 Direction	20
6.4.4.3 Mass	20
6.4.4.4 Speed	20
6.4.4.5 Velocity	20
6.5 Simulation.Encounter Struct Reference	20
6.5.1 Detailed Description	21
6.5.2 Property Documentation	21
6.5.2.1 Distance	21
6.5.2.2 NULL	21
6.5.2.3 Timestamp	21
6.5.2.4 WorldSnapshot	22
6.6 Simulation.World.Object Class Reference	22
6.6.1 Detailed Description	22
6.6.2 Constructor & Destructor Documentation	23
6.6.2.1 Object()	23
6.6.3 Member Function Documentation	23
6.6.3.1 CloneToWorld()	23
6.6.3.2 Step()	23
6.6.4 Property Documentation	24
6.6.4.1 ID	24
6.6.4.2 Location	24
6.7 Simulation.Program Class Reference	24
6.7.1 Detailed Description	25
6.7.2 Member Function Documentation	25
6.7.2.1 FindClosestEncounter()	25
6.7.3 Member Data Documentation	26
6.7.3.1 a1	26
6.7.3.2 a2	26
6.7.3.3 m1	26
6.7.3.4 m2	26
6.7.3.5 s1	26
6.7.3.6 s2	27
6.7.3.7 v1	27
6.7.3.8 v2	27
6.8 Simulation.Simulation Class Reference	27
6.8.1 Detailed Description	28

6.8.2 Constructor & Destructor Documentation	28
6.8.2.1 Simulation()	28
6.8.3 Member Function Documentation	28
6.8.3.1 DoStep()	28
6.8.3.2 Simulate()	29
6.8.3.3 SteppedEventHandler()	29
6.8.3.4 StopSimulation()	29
6.8.4 Member Data Documentation	29
6.8.4.1 world	29
6.8.5 Property Documentation	29
6.8.5.1 TimeSinceEpoch	29
6.8.6 Event Documentation	30
6.8.6.1 Stepped	30
6.9 Simulation.SimulationWatcher Class Reference	30
6.9.1 Detailed Description	30
6.9.2 Constructor & Destructor Documentation	31
6.9.2.1 SimulationWatcher()	31
6.9.3 Member Function Documentation	31
6.9.3.1 Dispose()	31
6.9.3.2 Simulation_Stepped()	31
6.10 Simulation.StepEventArgs Class Reference	32
6.10.1 Detailed Description	32
6.10.2 Property Documentation	32
6.10.2.1 DeltaTime	32
6.11 Simulation.SteppedEventArgs Class Reference	32
6.11.1 Detailed Description	33
6.11.2 Property Documentation	33
6.11.2.1 TimeSinceEpoch	33
6.11.2.2 WorldSnapshot	33
6.12 Simulation.World Class Reference	33
6.12.1 Detailed Description	34
6.12.2 Constructor & Destructor Documentation	34
6.12.2.1 World()	34
6.12.3 Member Function Documentation	34
6.12.3.1 Clone()	35
6.12.3.2 DoStep()	35
6.12.3.3 FindObject()	35
6.12.3.4 GetObjectByIndex()	35
6.12.3.5 StepEventHandler()	36
6.12.4 Event Documentation	36
6.12.4.1 Sten	36

7 File Documentation	37
7.1 ClosestEncounterWatcher.cs File Reference	37
7.2 ClosestEncounterWatcher.cs	37
7.3 ClosestEncounterWatcherSettings.cs File Reference	38
7.4 ClosestEncounterWatcherSettings.cs	38
7.5 DynamicWorldObject.cs File Reference	38
7.6 DynamicWorldObject.cs	39
7.7 Encounter.cs File Reference	39
7.8 Encounter.cs	40
7.9 obj/Debug/net5.0/.NETCoreApp,Version=v5.0.AssemblyAttributes.cs File Reference	40
7.10 .NETCoreApp,Version=v5.0.AssemblyAttributes.cs	40
7.11 obj/Debug/net5.0/Simulation.AssemblyInfo.cs File Reference	40
7.12 Simulation.AssemblyInfo.cs	40
7.13 Program.cs File Reference	41
7.14 Program.cs	41
7.15 Simulation.cs File Reference	42
7.16 Simulation.cs	42
7.17 SimulationWatcher.cs File Reference	43
7.18 SimulationWatcher.cs	43
7.19 StepEventArgs.cs File Reference	44
7.20 StepEventArgs.cs	44
7.21 SteppedEventArgs.cs File Reference	44
7.22 SteppedEventArgs.cs	44
7.23 StepSimulation.cs File Reference	45
7.24 StepSimulation.cs	45
7.25 World.cs File Reference	45
7.26 World on	16

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):								
Simulation	9							

2 Namespace Index

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Simulation.ClosestEncounterWatcherSettings		 										 	14
Simulation. Encounter		 										 	20
EventArgs													
Simulation.StepEventArgs									 			 	32
Simulation.SteppedEventArgs									 			 	32
ICloneable													
Simulation.World									 			 	33
IDisposable													
Simulation.SimulationWatcher									 			 	30
Simulation.ClosestEncounterWatcher .									 				11
Simulation.World.Object		 										 	22
Simulation.DynamicWorldPoint									 			 	16
Simulation.Program		 											24
Simulation.Simulation													
Simulation.ConstantStepSimulation									 			 	15

4 Hierarchical Index

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Simulation.ClosestEncounterWatcher	
A class that looks for a closest encounter of two objects in a simulation	11
Simulation.ClosestEncounterWatcherSettings	
Settings that define how a ClosestEncounterWatcher acts	14
Simulation.ConstantStepSimulation	
A simulation, that has a constant delay between each step.	15
Simulation.DynamicWorldPoint	
A point in a world that can move. That means this object is also defined by Velocity, current	
Acceleration and its Mass	16
Simulation. Encounter	
Struct that is used to describe encounter of two objects	20
Simulation.World.Object	
Abstract class used to define a world object	22
Simulation.Program	24
Simulation. Simulation	
A base class used to define how world is simulated	27
Simulation.SimulationWatcher	
Abstract class used to define so called watchers. Watchers are objects that can analyze a world	
every simulation step. For example a watcher can look for a closest encounter of two objects	30
Simulation.StepEventArgs	
Class that holds data about the current step	32
Simulation.SteppedEventArgs	
Class that holds data about a step that has been taken	32
Simulation.World	
An object used to hold world objects(see World.Object)	33

6 Class Index

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

ClosestEncounterWatcher.cs	37
ClosestEncounterWatcherSettings.cs	38
DynamicWorldObject.cs	38
Encounter.cs	39
Program.cs	11
Simulation.cs	12
SimulationWatcher.cs	13
StepEventArgs.cs	14
SteppedEventArgs.cs	14
StepSimulation.cs	15
World.cs	15
obj/Debug/net5.0/.NETCoreApp,Version=v5.0.AssemblyAttributes.cs	10
obj/Debug/net5.0/Simulation.AssemblyInfo.cs	10

8 File Index

Chapter 5

Namespace Documentation

5.1 Simulation Namespace Reference

Classes

· class ClosestEncounterWatcher

A class that looks for a closest encounter of two objects in a simulation

• struct ClosestEncounterWatcherSettings

Settings that define how a ClosestEncounterWatcher acts

class ConstantStepSimulation

A simulation, that has a constant delay between each step.

class DynamicWorldPoint

A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass

struct Encounter

Struct that is used to describe encounter of two objects

- · class Program
- · class Simulation

A base class used to define how world is simulated

• class SimulationWatcher

Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.

class StepEventArgs

Class that holds data about the current step

class SteppedEventArgs

Class that holds data about a step that has been taken

· class World

An object used to hold world objects(see World.Object)

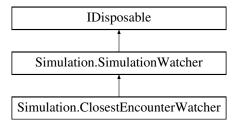
Chapter 6

Class Documentation

6.1 Simulation.ClosestEncounterWatcher Class Reference

A class that looks for a closest encounter of two objects in a simulation

Inheritance diagram for Simulation.ClosestEncounterWatcher:



Public Member Functions

- ClosestEncounterWatcher (Simulation simulation, World.Object obj1, World.Object obj2)
 - Creates an instance of ClosestEncounterWatcher
- ClosestEncounterWatcher (Simulation simulation, World.Object obj1, World.Object obj2, ShortestDistance
 —
 FinderSettings settings)

Creates an instance of ClosestEncounterWatcher

void ResetClosestEncounter ()

Discards the current closest encounter and starts looking for a new one

Static Public Attributes

static readonly ShortestDistanceFinderSettings DEFAULT_SETTINGS
 The default settings

Protected Member Functions

override void Simulation_Stepped (object sender, SteppedEventArgs e)
 Gets invoked after every step in the watched simulation

Properties

• Encounter CurrentClosestEncounter [get]

Latest recorded closest encounter

6.1.1 Detailed Description

A class that looks for a closest encounter of two objects in a simulation

Definition at line 8 of file ClosestEncounterWatcher.cs.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 ClosestEncounterWatcher() [1/2]

Creates an instance of ClosestEncounterWatcher

Parameters

simulation	The simulation on which the watcher should work
obj1	The first object which the watcher should watch
obj2	The second object which the watcher should watch

Definition at line 24 of file ClosestEncounterWatcher.cs.

6.1.2.2 ClosestEncounterWatcher() [2/2]

Creates an instance of ClosestEncounterWatcher

Parameters

simulation	The simulation on which the watcher should work
obj1	The first object which the watcher should watch
obj2	The second object which the watcher should watch
settings	Settings defining how the watcher should act

Definition at line 35 of file ClosestEncounterWatcher.cs.

6.1.3 Member Function Documentation

6.1.3.1 ResetClosestEncounter()

```
void Simulation.ClosestEncounterWatcher.ResetClosestEncounter ( )
```

Discards the current closest encounter and starts looking for a new one

6.1.3.2 Simulation_Stepped()

Gets invoked after every step in the watched simulation

Parameters

sender	Instance of the object that invoked the step
е	Instance of SteppedEventArgs describing the state of the simulated world after the step

Implements Simulation.SimulationWatcher.

Definition at line 58 of file ClosestEncounterWatcher.cs.

6.1.4 Member Data Documentation

6.1.4.1 DEFAULT_SETTINGS

readonly ShortestDistanceFinderSettings Simulation.ClosestEncounterWatcher.DEFAULT_SETTINGS [static]

Initial value:

The default settings

Definition at line 13 of file ClosestEncounterWatcher.cs.

6.1.5 Property Documentation

6.1.5.1 CurrentClosestEncounter

Encounter Simulation.ClosestEncounterWatcher.CurrentClosestEncounter [get]

Latest recorded closest encounter

Definition at line 45 of file ClosestEncounterWatcher.cs.

The documentation for this class was generated from the following file:

ClosestEncounterWatcher.cs

6.2 Simulation.ClosestEncounterWatcherSettings Struct Reference

Settings that define how a ClosestEncounterWatcher acts

Properties

• bool AutoStopSimulation [getset]

True, if the watcher should stop the simulation when the objects start to move away

6.2.1 Detailed Description

Settings that define how a ClosestEncounterWatcher acts

Definition at line 6 of file ClosestEncounterWatcherSettings.cs.

6.2.2 Property Documentation

6.2.2.1 AutoStopSimulation

bool Simulation.ClosestEncounterWatcherSettings.AutoStopSimulation [get], [set]

True, if the watcher should stop the simulation when the objects start to move away

Definition at line 11 of file ClosestEncounterWatcherSettings.cs.

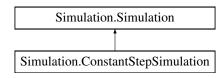
The documentation for this struct was generated from the following file:

ClosestEncounterWatcherSettings.cs

6.3 Simulation.ConstantStepSimulation Class Reference

A simulation, that has a constant delay between each step.

Inheritance diagram for Simulation. ConstantStepSimulation:



Public Member Functions

• ConstantStepSimulation (World world, TimeSpan stepLength, TimeSpan simulationLength)

Creagtes an instance of ConstantStepSimulation

Protected Member Functions

• override void DoStep ()

Additional Inherited Members

6.3.1 Detailed Description

A simulation, that has a constant delay between each step.

Definition at line 8 of file StepSimulation.cs.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ConstantStepSimulation()

Creagtes an instance of ConstantStepSimulation

Parameters

world	A world that will be simulated
stepLength	A time between each step (smaller value = more precision)
simulationLength	A total length of simulation

Definition at line 16 of file StepSimulation.cs.

6.3.3 Member Function Documentation

6.3.3.1 DoStep()

override void Simulation.ConstantStepSimulation.DoStep () [protected], [virtual]

Implements Simulation. Simulation.

Definition at line 25 of file StepSimulation.cs.

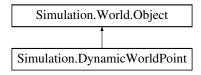
The documentation for this class was generated from the following file:

• StepSimulation.cs

6.4 Simulation.DynamicWorldPoint Class Reference

A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass

Inheritance diagram for Simulation. Dynamic World Point:



Public Member Functions

• DynamicWorldPoint (World world, Vector2 location, float mass)

Creates an instance of DynamicWorldPoint and binds it with a specific World

DynamicWorldPoint (World world, Vector2 location, float mass, Vector2 velocity)

Creates an instance of DynamicWorldPoint with a specific starting velocity and binds it with a specific World

void StartApplyingForce (Vector2 force)

Starts applying a force to the point

• void StopApplyingForce ()

Stops all forces, that are currently being applied on the point

override World.Object CloneToWorld (World world)

Clones this world object to a different instance of World and binds it to that world

Protected Member Functions

override void Step (object sender, StepEventArgs e)

A method which is called when a step is being taken

Properties

• Vector2 Acceleration [get]

A vector defining current acceleration and direction, at which the point is accelerating

• Vector2 Velocity [getset]

A vector defining the points speed and direction in which it is travelling

• float Mass [get]

A mass of the point

• float Speed [get]

A speed at which the point is travelling

• double Direction [get]

A direction at wihch the point is travelling

6.4.1 Detailed Description

A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass

Definition at line 9 of file DynamicWorldObject.cs.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 DynamicWorldPoint() [1/2]

Creates an instance of DynamicWorldPoint and binds it with a specific World

Parameters

world	An instance of World in which the point exists and is bound to	
location	The location in relation to the origin in the World that the point is bound	
mass	The mass of the point	

Definition at line 17 of file DynamicWorldObject.cs.

6.4.2.2 DynamicWorldPoint() [2/2]

```
\label{thm:condition} Simulation. Dynamic World Point. Dynamic World Point \ ( \\ World \ world,
```

```
Vector2 location,
float mass,
Vector2 velocity )
```

Creates an instance of DynamicWorldPoint with a specific starting velocity and binds it with a specific World

Parameters

world	An instance of World in which the point exists and is bound to		
location	The location in relation to the origin in the World that the point is bound to		
mass	The mass of the point		
velocity A vector defining the points speed and direction in which it is travelli			

Definition at line 28 of file DynamicWorldObject.cs.

6.4.3 Member Function Documentation

6.4.3.1 CloneToWorld()

Clones this world object to a different instance of World and binds it to that world

Parameters

world	An instance of World the object is being cloned to
-------	--

Returns

A cloned instance of World.Object

Implements Simulation.World.Object.

Definition at line 73 of file DynamicWorldObject.cs.

6.4.3.2 StartApplyingForce()

```
void Simulation.DynamicWorldPoint.StartApplyingForce ( \label{eq:point} \mbox{Vector2 force )}
```

Starts applying a force to the point

Parameters

ctor defining a force and a direction in which it is applied
--

Definition at line 60 of file DynamicWorldObject.cs.

6.4.3.3 Step()

A method which is called when a step is being taken

Parameters

sender	Object that invoked the step
е	Instance of StepEventArgs describing the step

Implements Simulation. World. Object.

Definition at line 80 of file DynamicWorldObject.cs.

6.4.3.4 StopApplyingForce()

```
void Simulation.DynamicWorldPoint.StopApplyingForce ( )
```

Stops all forces, that are currently being applied on the point

Definition at line 68 of file DynamicWorldObject.cs.

6.4.4 Property Documentation

6.4.4.1 Acceleration

```
Vector2 Simulation.DynamicWorldPoint.Acceleration [get]
```

A vector defining current acceleration and direction, at which the point is accelerating

Definition at line 37 of file DynamicWorldObject.cs.

6.4.4.2 Direction

```
double Simulation.DynamicWorldPoint.Direction [get]
```

A direction at wihch the point is travelling

Definition at line 54 of file DynamicWorldObject.cs.

6.4.4.3 Mass

```
float Simulation.DynamicWorldPoint.Mass [get]
```

A mass of the point

Definition at line 45 of file DynamicWorldObject.cs.

6.4.4.4 Speed

```
float Simulation.DynamicWorldPoint.Speed [get]
```

A speed at which the point is travelling

Definition at line 50 of file DynamicWorldObject.cs.

6.4.4.5 Velocity

```
Vector2 Simulation.DynamicWorldPoint.Velocity [get], [set]
```

A vector defining the points speed and direction in which it is travelling

Definition at line 41 of file DynamicWorldObject.cs.

The documentation for this class was generated from the following file:

• DynamicWorldObject.cs

6.5 Simulation. Encounter Struct Reference

Struct that is used to describe encounter of two objects

Properties

• static Encounter NULL [get]

Instance of Encounter used to describe no encounter

• float Distance [getset]

Distance between the two objects during the encounter

• TimeSpan Timestamp [getset]

Timestamp at which the encounter occured

• World WorldSnapshot [getset]

A snapshot of the word at the time the encounter happened

6.5.1 Detailed Description

Struct that is used to describe encounter of two objects

Definition at line 8 of file Encounter.cs.

6.5.2 Property Documentation

6.5.2.1 Distance

```
float Simulation.Encounter.Distance [get], [set]
```

Distance between the two objects during the encounter

Definition at line 23 of file Encounter.cs.

6.5.2.2 NULL

```
Encounter Simulation.Encounter.NULL [static], [get]
```

Instance of Encounter used to describe no encounter

Definition at line 13 of file Encounter.cs.

6.5.2.3 Timestamp

```
TimeSpan Simulation.Encounter.Timestamp [get], [set]
```

Timestamp at which the encounter occured

Definition at line 27 of file Encounter.cs.

6.5.2.4 WorldSnapshot

```
World Simulation.Encounter.WorldSnapshot [get], [set]
```

A snapshot of the word at the time the encounter happened

Definition at line 31 of file Encounter.cs.

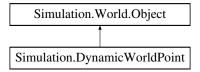
The documentation for this struct was generated from the following file:

· Encounter.cs

6.6 Simulation.World.Object Class Reference

Abstract class used to define a world object

Inheritance diagram for Simulation. World. Object:



Public Member Functions

· Object (World world, Vector2 location)

Creates instance of a world object

abstract Object CloneToWorld (World world)

Clones this world object to a different instance of World and binds it to that world

Protected Member Functions

• abstract void Step (object sender, StepEventArgs e)

A method which is called when a step is being taken

Properties

• Vector2 Location [getset]

The location in relation to the origin

• Guid ID [get]

A unique ID od the world object

6.6.1 Detailed Description

Abstract class used to define a world object

Definition at line 66 of file World.cs.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Object()

Creates instance of a world object

Parameters

world	An instance of World in which the point exists and is bound	
location	The location in relation to the origin	

Definition at line 73 of file World.cs.

6.6.3 Member Function Documentation

6.6.3.1 CloneToWorld()

Clones this world object to a different instance of World and binds it to that world

Parameters

world An instance of World the object is being cloned to

Returns

A cloned instance of World.Object

Implemented in Simulation.DynamicWorldPoint.

6.6.3.2 Step()

A method which is called when a step is being taken

Parameters

sender	Object that invoked the step
е	Instance of StepEventArgs describing the step

Implemented in Simulation.DynamicWorldPoint.

6.6.4 Property Documentation

6.6.4.1 ID

Guid Simulation.World.Object.ID [get]

A unique ID od the world object

Definition at line 91 of file World.cs.

6.6.4.2 Location

Vector2 Simulation.World.Object.Location [get], [set]

The location in relation to the origin

Definition at line 84 of file World.cs.

The documentation for this class was generated from the following file:

• World.cs

6.7 Simulation.Program Class Reference

Static Public Member Functions

• static Encounter FindClosestEncounter (Simulation simulation, DynamicWorldPoint obj1, DynamicWorldPoint obj2, bool reportTime=false)

Runs the simulation and records closest encounter of two objects

Static Public Attributes

• const float s1 = 15

Distance of Point1 from the origin of the right angle

• const float s2 = 10

Distance of Point2 from the origin of the right angle

const float v1 = 15

The size of velocity vector used by Point1 that is heading to the origin of the right angle

const float v2 = 20

The size of velocity vector used by Point2 that is heading to the origin of the right angle

• const float m1 = 10

The mass of Point1

const float m2 = 20

The mass of Point2

• const float a1 = 2

The acceleration used by Point1 (used only by the acceleration simulation)

• const float a2 = 1

The acceleration used by Point2 (used only by the acceleration simulation)

6.7.1 Detailed Description

Definition at line 9 of file Program.cs.

6.7.2 Member Function Documentation

6.7.2.1 FindClosestEncounter()

Runs the simulation and records closest encounter of two objects

Parameters

simulation	The simulation in which the objects are simulated
obj1	First object that should be watched
obj2	Second object that should be watched

Returns

Instance of Encounter describing the closest encounter

Definition at line 95 of file Program.cs.

6.7.3 Member Data Documentation

6.7.3.1 a1

```
const float Simulation.Program.a1 = 2 [static]
```

The acceleration used by Point1 (used only by the acceleration simulation)

Definition at line 52 of file Program.cs.

6.7.3.2 a2

```
const float Simulation.Program.a2 = 1 [static]
```

The acceleration used by Point2 (used only by the acceleration simulation)

Definition at line 56 of file Program.cs.

6.7.3.3 m1

```
const float Simulation.Program.m1 = 10 [static]
```

The mass of Point1

Definition at line 43 of file Program.cs.

6.7.3.4 m2

```
const float Simulation.Program.m2 = 20 [static]
```

The mass of Point2

Definition at line 47 of file Program.cs.

6.7.3.5 s1

```
const float Simulation.Program.s1 = 15 [static]
```

Distance of Point1 from the origin of the right angle

Definition at line 25 of file Program.cs.

6.7.3.6 s2

```
const float Simulation.Program.s2 = 10 [static]
```

Distance of Point2 from the origin of the right angle

Definition at line 29 of file Program.cs.

6.7.3.7 v1

```
const float Simulation.Program.v1 = 15 [static]
```

The size of velocity vector used by Point1 that is heading to the origin of the right angle

Definition at line 34 of file Program.cs.

6.7.3.8 v2

```
const float Simulation.Program.v2 = 20 [static]
```

The size of velocity vector used by Point2 that is heading to the origin of the right angle

Definition at line 38 of file Program.cs.

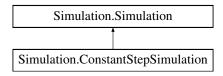
The documentation for this class was generated from the following file:

• Program.cs

6.8 Simulation.Simulation Class Reference

A base class used to define how world is simulated

Inheritance diagram for Simulation. Simulation:



Public Member Functions

- delegate void SteppedEventHandler (object sender, SteppedEventArgs e)
 Handler used to handle a stepped event
- Simulation (World world)
- void StopSimulation ()
- void Simulate ()

Protected Member Functions

• abstract void DoStep ()

Protected Attributes

World world

Properties

• TimeSpan TimeSinceEpoch [getprotected set]

Events

• SteppedEventHandler Stepped

6.8.1 Detailed Description

A base class used to define how world is simulated

Definition at line 8 of file Simulation.cs.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Simulation()

```
Simulation.Simulation ( World\ world )
```

Definition at line 17 of file Simulation.cs.

6.8.3 Member Function Documentation

6.8.3.1 DoStep()

```
abstract void Simulation.Simulation.DoStep ( ) [protected], [pure virtual]
```

Implemented in Simulation. ConstantStepSimulation.

6.8.3.2 Simulate()

```
void Simulation.Simulation.Simulate ( )
```

Definition at line 32 of file Simulation.cs.

6.8.3.3 SteppedEventHandler()

Handler used to handle a stepped event

Parameters

sender	Object that invoked the step	
е	Instance of SteppedEventArgs describing the state of the simulated world after a step	

6.8.3.4 StopSimulation()

```
void Simulation.Simulation.StopSimulation ( )
```

6.8.4 Member Data Documentation

6.8.4.1 world

```
World Simulation.Simulation.world [protected]
```

Definition at line 24 of file Simulation.cs.

6.8.5 Property Documentation

6.8.5.1 TimeSinceEpoch

```
TimeSpan Simulation.Simulation.TimeSinceEpoch [get], [protected set]
```

Definition at line 30 of file Simulation.cs.

6.8.6 Event Documentation

6.8.6.1 Stepped

SteppedEventHandler Simulation.Simulation.Stepped

Definition at line 22 of file Simulation.cs.

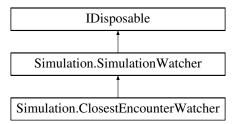
The documentation for this class was generated from the following file:

· Simulation.cs

6.9 Simulation.SimulationWatcher Class Reference

Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.

Inheritance diagram for Simulation.SimulationWatcher:



Public Member Functions

• SimulationWatcher (Simulation simulation)

Creates an instance of watcher, that analyzes a specific simulation

· void Dispose ()

Releases the watcher from the simulation

Protected Member Functions

abstract void Simulation_Stepped (object sender, SteppedEventArgs e)
 Gets invoked after every step in the watched simulation

6.9.1 Detailed Description

Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.

Definition at line 9 of file SimulationWatcher.cs.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 SimulationWatcher()

```
Simulation.SimulationWatcher.SimulationWatcher ( Simulation \ simulation )
```

Creates an instance of watcher, that analyzes a specific simulation

Parameters

simulatio	n	Instance of Simulation, that the watcher watches
-----------	---	--

Definition at line 15 of file SimulationWatcher.cs.

6.9.3 Member Function Documentation

6.9.3.1 Dispose()

```
void Simulation.SimulationWatcher.Dispose ( )
```

Releases the watcher from the simulation

6.9.3.2 Simulation_Stepped()

Gets invoked after every step in the watched simulation

Parameters

sender	Instance of the object that invoked the step	
е	Instance of SteppedEventArgs describing the state of the simulated world after the step	

Implemented in Simulation.ClosestEncounterWatcher.

The documentation for this class was generated from the following file:

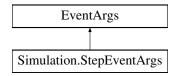
• SimulationWatcher.cs

32 Class Documentation

6.10 Simulation.StepEventArgs Class Reference

Class that holds data about the current step

Inheritance diagram for Simulation.StepEventArgs:



Properties

• TimeSpan DeltaTime [getset]

Time, between the current step and the previous step

6.10.1 Detailed Description

Class that holds data about the current step

Definition at line 8 of file StepEventArgs.cs.

6.10.2 Property Documentation

6.10.2.1 DeltaTime

TimeSpan Simulation.StepEventArgs.DeltaTime [get], [set]

Time, between the current step and the previous step

Definition at line 13 of file StepEventArgs.cs.

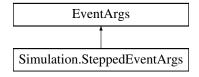
The documentation for this class was generated from the following file:

StepEventArgs.cs

6.11 Simulation.SteppedEventArgs Class Reference

Class that holds data about a step that has been taken

Inheritance diagram for Simulation.SteppedEventArgs:



Properties

• World WorldSnapshot [getset]

A clone of the world after the step was taken

• TimeSpan TimeSinceEpoch [getset]

A since the world has been created in the world time

6.11.1 Detailed Description

Class that holds data about a step that has been taken

Definition at line 8 of file SteppedEventArgs.cs.

6.11.2 Property Documentation

6.11.2.1 TimeSinceEpoch

TimeSpan Simulation.SteppedEventArgs.TimeSinceEpoch [get], [set]

A since the world has been created in the world time

Definition at line 17 of file SteppedEventArgs.cs.

6.11.2.2 WorldSnapshot

World Simulation.SteppedEventArgs.WorldSnapshot [get], [set]

A clone of the world after the step was taken

Definition at line 13 of file SteppedEventArgs.cs.

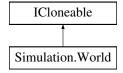
The documentation for this class was generated from the following file:

• SteppedEventArgs.cs

6.12 Simulation. World Class Reference

An object used to hold world objects(see World.Object)

Inheritance diagram for Simulation. World:



34 Class Documentation

Classes

· class Object

Abstract class used to define a world object

Public Member Functions

• delegate void StepEventHandler (object sender, StepEventArgs e)

Handler used to handle a step event

• World ()

Creates a new world

void DoStep (TimeSpan deltaTime)

Takes a simulation step

- object Clone ()
- Object FindObject (Guid id)

Find a objects by its ID

Object GetObjectByIndex (int index)

Gets object by its index. Indexes are being asigned by order the objects were binded to the world.

Events

• StepEventHandler Step

Event, which gets invoked when a new step is being taken

6.12.1 Detailed Description

An object used to hold world objects(see World.Object)

Definition at line 10 of file World.cs.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 World()

```
Simulation.World.World ( ) \,
```

Creates a new world

Definition at line 22 of file World.cs.

6.12.3 Member Function Documentation

6.12.3.1 Clone()

```
object Simulation.World.Clone ( )
```

Definition at line 43 of file World.cs.

6.12.3.2 DoStep()

```
void Simulation.World.DoStep ( {\tt TimeSpan} \ \textit{deltaTime} \ )
```

Takes a simulation step

Parameters

Definition at line 38 of file World.cs.

6.12.3.3 FindObject()

Find a objects by its ID

Parameters

id ID of the object

Returns

An instance of the object from this world

6.12.3.4 GetObjectByIndex()

```
Object Simulation.World.GetObjectByIndex (
    int index )
```

Gets object by its index. Indexes are being asigned by order the objects were binded to the world.

36 Class Documentation

Parameters

index

Returns

6.12.3.5 StepEventHandler()

```
delegate void Simulation.World.StepEventHandler ( object \ sender, \\ StepEventArgs \ e \ )
```

Handler used to handle a step event

Parameters

sender	Object that invoked the step
е	Instance of StepEventArgs describing the step

6.12.4 Event Documentation

6.12.4.1 Step

StepEventHandler Simulation.World.Step

Event, which gets invoked when a new step is being taken

Definition at line 30 of file World.cs.

The documentation for this class was generated from the following file:

• World.cs

Chapter 7

File Documentation

7.1 ClosestEncounterWatcher.cs File Reference

Classes

· class Simulation.ClosestEncounterWatcher

A class that looks for a closest encounter of two objects in a simulation

Namespaces

namespace Simulation

7.2 ClosestEncounterWatcher.cs

```
00001 using System;
00002
00003 namespace Simulation
00004 {
80000
          public class ClosestEncounterWatcher : SimulationWatcher
00009
              public static readonly ShortestDistanceFinderSettings DEFAULT_SETTINGS = new
ShortestDistanceFinderSettings()
00015
                  {\tt AutoStopSimulation} \, = \, {\tt false}
00016
             };
00017
00024
              public ClosestEncounterWatcher(Simulation simulation, World.Object obj1, World.Object obj2) :
      this (simulation, obj1, obj2, DEFAULT_SETTINGS)
00025
00026
00027
             public ClosestEncounterWatcher(Simulation simulation, World.Object obj1, World.Object obj2,
00035
      ShortestDistanceFinderSettings settings) : base(simulation)
00036
            {
00037
                  this.settings = settings;
00038
                  this.obj1 = obj1;
                 this.obj2 = obj2;
00039
00040
00041
00045
             public Encounter CurrentClosestEncounter => closestEncounter;
00046
             private Encounter closestEncounter = Encounter.NULL;
00047
00048
             private ShortestDistanceFinderSettings settings;
00049
             private float lastDistance = float.PositiveInfinity;
00050
             private readonly World.Object obj1;
00051
             private readonly World. Object obj2;
```

```
00052
              public void ResetClosestEncounter() => closestEncounter = Encounter.NULL;
00057
              protected override void Simulation_Stepped(object sender, SteppedEventArgs e)
00058
00059
00060
                  float distance = (obj1.Location - obj2.Location).Length();
00061
00062
                   // If the distance between objects is closer than anytime before -> record the encounter
00063
                  if (distance < closestEncounter.Distance)</pre>
00064
                       closestEncounter = new Encounter()
00065
00066
                           Distance = distance,
                           Timestamp = e.TimeSinceEpoch,
00067
00068
                           WorldSnapshot = e.WorldSnapshot
00069
00070
                  // When objects are starting to move away and settings allow it \rightarrow stop the simulation
00071
00072
                  if (settings.AutoStopSimulation && distance > lastDistance)
                      simulation.StopSimulation();
00074
00075
                  lastDistance = distance;
00076
00077
          }
00078 }
```

7.3 ClosestEncounterWatcherSettings.cs File Reference

Classes

struct Simulation.ClosestEncounterWatcherSettings
 Settings that define how a ClosestEncounterWatcher acts

Namespaces

namespace Simulation

7.4 ClosestEncounterWatcherSettings.cs

```
Go to the documentation of this file.
```

7.5 DynamicWorldObject.cs File Reference

Classes

· class Simulation.DynamicWorldPoint

A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass

Namespaces

namespace Simulation

7.6 DynamicWorldObject.cs

Go to the documentation of this file.

```
00001 using System;
00002 using System.Numerics;
00003
00004 namespace Simulation
00005 {
00009
          public class DynamicWorldPoint : World.Object
00010
00017
               public DynamicWorldPoint(World world, Vector2 location, float mass) : this(world, location,
       mass, new Vector2(0))
00018
00019
00020
00028
               public DynamicWorldPoint(World world, Vector2 location, float mass, Vector2 velocity) :
      base(world, location)
00029
             {
00030
                   Velocity = velocity;
00031
00032
00033
              public Vector2 Acceleration { get; private set; }
public Vector2 Velocity { get; set; }
public float Mass { get; }
00037
00041
00045
00046
00050
               public float Speed => Velocity.Length();
00054
               public double Direction => Math.Atan2(Velocity.Y, Velocity.X);
00055
00060
               public void StartApplyingForce(Vector2 force)
00061
00062
                   Acceleration += force / Mass;
00063
00064
00068
               public void StopApplyingForce()
00069
00070
                   Acceleration = new Vector2(0):
00071
00072
00073
               public override World.Object CloneToWorld(World world)
00074
00075
                   DynamicWorldPoint clone = new DynamicWorldPoint(world, Location, Mass, Velocity);
00076
                   clone.Acceleration = Acceleration;
                   return clone;
00078
              }
00079
00080
               protected override void Step(object sender, StepEventArgs e)
00081
00082
                   Location += Velocity * (float)e.DeltaTime.TotalSeconds;
                   Velocity += Acceleration * (float)e.DeltaTime.TotalSeconds;
00083
00085
00086 }
```

7.7 Encounter.cs File Reference

Classes

· struct Simulation. Encounter

Struct that is used to describe encounter of two objects

Namespaces

namespace Simulation

7.8 Encounter.cs

Go to the documentation of this file.

```
00001 using System;
00002
00003 namespace Simulation
00004 {
80000
           public struct Encounter
00009
00013
               public static Encounter NULL => new Encounter()
00014
00015
                    Distance = float.PositiveInfinity,
                    Timestamp = TimeSpan.FromSeconds(-1),
00016
                   WorldSnapshot = null
00018
00019
              public float Distance { get; set; }
00023
              public TimeSpan Timestamp { get; set; }
public World WorldSnapshot { get; set; }
00027
00032
00033 }
```

7.9 obj/Debug/net5.0/.NETCoreApp,Version=v5.0.AssemblyAttributes.cs File Reference

7.10 .NETCoreApp,Version=v5.0.AssemblyAttributes.cs

Go to the documentation of this file.

7.11 obj/Debug/net5.0/Simulation.AssemblyInfo.cs File Reference

7.12 Simulation. Assembly Info.cs

```
00002 // <auto-generated>
00003 //
             This code was generated by a tool.
00004 //
            Runtime Version: 4.0.30319.42000
00005 //
00006 //
            Changes to this file may cause incorrect behavior and will be lost if
00007 //
             the code is regenerated.
00008 // </auto-generated>
00009 //---
00010
00011 using System;
00012 using System.Reflection;
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("Simulation")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("Simulation")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("Simulation")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generated by the MSBuild WriteCodeFragment class.
00023
```

7.13 Program.cs File Reference

Classes

· class Simulation.Program

Namespaces

· namespace Simulation

7.14 Program.cs

```
Go to the documentation of this file.
00001 using System;
00002 using System.Collections.Generic;
00003 using System.IO;
00004 using System.Numerics;
00005 using System. Threading. Tasks;
00006
00007 namespace Simulation
00008 {
00009
          class Program
00010
          {
00011
00012
               * POINT1
00013
00014
00015
00016
00017
00018
00019
                       ---- < POINT2
00020
00021
00025
              public const float s1 = 15;
00029
              public const float s2 = 10;
00030
00034
              public const float v1 = 15;
00038
              public const float v2 = 20;
00039
00043
              public const float m1 = 10;
00047
              public const float m2 = 20;
00048
00052
              public const float a1 = 2;
00056
              public const float a2 = 1;
00057
00058
              static void Main(string[] args)
00059
00060
                   Console.WriteLine("Creating worlds...");
00061
                   // Create seperate worlds for each simulation
00062
                  World constVelWorld = new World();
00063
                  var constPoint1 = new DynamicWorldPoint(constVelWorld, new Vector2(0, s1), m1, new
       Vector2(0, -v1));
00064
                  var constPoint2 = new DynamicWorldPoint(constVelWorld, new Vector2(s2, 0), m2, new
       Vector2(-v2, 0));
00065
00066
                  World accelVelWorld = new World();
00067
                  var accelPoint1 = new DynamicWorldPoint(accelVelWorld, new Vector2(0, s1), m1, new
       Vector2(0, -v1));
                  accelPoint1.StartApplyingForce(new Vector2(0, -(a1 * m1)));
00068
                  var accelPoint2 = new DynamicWorldPoint (accelVelWorld, new Vector2(s2, 0), m2, new
00069
       Vector2(-v2, 0));
00070
                  accelPoint2.StartApplyingForce(new Vector2(-(a2 * m2), 0));
00071
                  Console.WriteLine("Creating simulations...");
Simulation constSimulation = new StepSimulation(constVelWorld,
00072
00073
       TimeSpan.FromMilliseconds(1), TimeSpan.FromSeconds(10));
00074
                  Simulation accelSimulation = new StepSimulation(accelVelWorld,
       TimeSpan.FromMilliseconds(1), TimeSpan.FromSeconds(10));
00075
00076
                   // Run the simulations
                  Console.WriteLine("Constant speed simulation:");
00077
00078
                  var closestConstEncounter = FindClosestEncounter(constSimulation, constPoint1,
       constPoint2, true);
```

```
00079
                 Console.WriteLine("Constant speed simulation:");
                 var closestAccelEncounter = FindClosestEncounter(accelSimulation, accelPoint1,
00080
       accelPoint2, true);
00081
00082
                 // Print the results
                 Console.WriteLine($"Closest encounter with constnant speed happend {
00083
      closestConstEncounter.Timestamp.TotalSeconds.ToString("0.000") } seconds after the epoch. Points were
       { closestConstEncounter.Distance }m apart from eachother.");
00084
                 {\tt Console.WriteLine}\,(\$"{\tt Closest \ encounter \ when \ accelerating \ happend}\ \{
      { closestAccelEncounter.Distance }m apart from eachother.");
                 Console.ReadKey(true);
00085
00086
00087
00095
             public static Encounter FindClosestEncounter(Simulation simulation, DynamicWorldPoint obj1,
      DynamicWorldPoint obj2, bool reportTime = false)
00096
             {
00097
                 Encounter closest;
00098
                 using (var watcher = new ClosestEncounterWatcher(simulation, obj1, obj2))
00099
00100
                     Console.WriteLine("Simulating...");
00101
                     Console.Write("Seconds Processed: ");
                     if (reportTime)
00102
00103
                         simulation. Stepped += Simulation ReportTime;
00104
00105
                     simulation.Simulate();
00106
                     Console.WriteLine();
00107
                     closest = watcher.CurrentClosestEncounter;
00108
                 }
00109
00110
                 if (reportTime)
00111
                     simulation.Stepped -= Simulation_ReportTime;
00112
00113
                 return closest;
00114
                 static void Simulation_ReportTime(object sender, SteppedEventArgs e)
00115
00116
00117
                     int x = Console.CursorLeft, y = Console.CursorTop;
00118
                     Console.Write(e.TimeSinceEpoch.TotalSeconds.ToString("0.000"));
00119
                     Console.SetCursorPosition(x, y);
00120
00121
             }
00122
         }
00123 }
```

7.15 Simulation.cs File Reference

Classes

· class Simulation.Simulation

A base class used to define how world is simulated

Namespaces

namespace Simulation

7.16 Simulation.cs

```
00001 using System;
00002
00003 namespace Simulation
00004 {
00008    public abstract class Simulation
00009    {
00015         public delegate void SteppedEventHandler(object sender, SteppedEventArgs e);
00016
00017         public Simulation(World world)
00018         {
```

```
00019
                  this.world = world;
00020
00021
00022
              public event SteppedEventHandler Stepped;
00023
00024
              protected World world:
00025
00026
              private bool cancellationRequested = false;
00027
00028
              public void StopSimulation() => cancellationRequested = true;
00029
00030
              public TimeSpan TimeSinceEpoch { get; protected set; }
00031
00032
              public void Simulate()
00033
00034
                   // First step (before updating any world objects)
00035
                  HandleStepped();
00036
00037
                  while (!cancellationRequested)
00038
                  {
00039
                      DoStep();
00040
                      HandleStepped();
00041
00042
00043
00044
              private void HandleStepped()
00045
00046
                  Stepped?.Invoke(this, new SteppedEventArgs()
00047
                      WorldSnapshot = world.Clone() as World,
00048
00049
                      TimeSinceEpoch = TimeSinceEpoch
00050
00051
00052
00053
              protected abstract void DoStep();
00054
          }
00055 }
```

7.17 SimulationWatcher.cs File Reference

Classes

· class Simulation.SimulationWatcher

Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.

Namespaces

• namespace Simulation

7.18 SimulationWatcher.cs

```
00001 using System;
00002 namespace Simulation
00003 {
00009
          public abstract class SimulationWatcher: IDisposable
00010
              public SimulationWatcher(Simulation simulation)
00016
                  simulation.Stepped += Simulation_Stepped;
00017
00018
                  this.simulation = simulation;
00019
00020
00024
              private readonly Simulation simulation;
00025
00031
              protected abstract void Simulation_Stepped(object sender, SteppedEventArgs e);
00032
00036
              public void Dispose() => simulation. Stepped -= Simulation Stepped:
00037
          }
00038 }
```

7.19 StepEventArgs.cs File Reference

Classes

• class Simulation.StepEventArgs

Class that holds data about the current step

Namespaces

· namespace Simulation

7.20 StepEventArgs.cs

Go to the documentation of this file.

7.21 SteppedEventArgs.cs File Reference

Classes

• class Simulation.SteppedEventArgs

Class that holds data about a step that has been taken

Namespaces

• namespace Simulation

7.22 SteppedEventArgs.cs

7.23 StepSimulation.cs File Reference

Classes

· class Simulation.ConstantStepSimulation

A simulation, that has a constant delay between each step.

Namespaces

· namespace Simulation

7.24 StepSimulation.cs

```
Go to the documentation of this file.
```

```
00001 using System;
00002
00003 namespace Simulation
00004 {
00008
          public class ConstantStepSimulation : Simulation
00009
               \verb|public ConstantStepSimulation(World world, TimeSpan stepLength, TimeSpan simulationLength)|:
      base(world)
00017
             {
00018
                   this.stepLength = stepLength;
00019
                   this.simulationLength = simulationLength;
00020
00021
              private readonly TimeSpan stepLength;
private readonly TimeSpan simulationLength;
00022
00023
00024
00025
              protected override void DoStep()
00026
00027
                   world.DoStep(stepLength);
00028
                   TimeSinceEpoch += stepLength;
00029
00030
                  if (TimeSinceEpoch > simulationLength)
00031
                       StopSimulation();
00032
00033
          }
00034 }
```

7.25 World.cs File Reference

Classes

· class Simulation.World

An object used to hold world objects(see World.Object)

• class Simulation.World.Object

Abstract class used to define a world object

Namespaces

• namespace Simulation

7.26 World.cs

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Numerics;
00004
00005 namespace Simulation
00006 {
          public class World : ICloneable
00010
00011
00017
              public delegate void StepEventHandler(object sender, StepEventArgs e);
00018
              public World()
00023
00024
                  WorldObjects = new List<Object>();
00025
00026
00030
              public event StepEventHandler Step;
00031
00032
              private readonly List<Object> WorldObjects;
00033
              public void DoStep(TimeSpan deltaTime)
00038
00039
00040
                   Step?.Invoke(this, new StepEventArgs() { DeltaTime = deltaTime });
00041
00042
00043
              public object Clone()
00044
                  World clone = new World();
00045
00046
                  WorldObjects.ForEach(obj => obj.CloneToWorld(clone));
00047
                  return clone;
00048
00049
00055
00061
               public Object FindObject(Guid id) => WorldObjects.Find(x => x.ID.Equals(id));
              public Object GetObjectByIndex(int index) => WorldObjects[index];
00062
00066
              public abstract class Object
00067
00073
                   public Object (World world, Vector2 location)
00074
                       world.Step += Step;
00075
00076
                       world.WorldObjects.Add(this);
00077
                       Location = location;
00078
                       id = Guid.NewGuid();
00079
08000
                  public Vector2 Location { get; set; }
00084
00085
00086
                  private Guid id;
00087
00091
                  public Guid ID => id;
00092
00098
                  protected abstract void Step(object sender, StepEventArgs e);
00099
00105
                  public abstract Object CloneToWorld(World world);
00106
              }
00107
00108 }
```