Die EDU-Klassen der  4

Methoden der Super-Klasse EduActor

animiereGerade(double sekunden, double zX, double zY, boolean loop)

Bewegt den Actor auf einer Strecke.

sekunden: *So lange dauert die Animation bis zum Zielpunkt (zX|zY)*
zX , zY: *Koordinaten des Punkts zu dem hin animiert werden soll*
loop: *false = einmalige Animation; true = dauerhaft hin und her*

animiereKreis(double sekunden, double mX, double mY, boolean uhrzeigersinn, boolean rotation)

Bewegt den Actor in einem Kreis um einen Angegebenen Mittelpunkt.

sekunden: *So lange dauert ein Umlauf*
mX , mY: *Mittelpunkt des Kreises der Animation*
uhrzeigersinn: *true = im Uhrzeigersinn; false = entgegen*
rotation: *false = einmal; true = endlos*

animiereTransparenz(double zeitInSekunden, double nachTransparenz)

Animiert flüssig die Transparenz dieses Actors von einem bestimmten Wert zu einem bestimmten Wert.

ZeitInSekunden: *So lange dauert der Übergang*
nachTransparenz: *zu diesem Transparenzwert führt der Übergang*

beinhaltetPunkt(double pX, double pY) : boolean

Gibt an, ob der EduActor einen bestimmten Punkt enthält.

pX , pY: *Die Koordinaten des Punkts*

drehen(double grad)

Dreht den EduActor um einen bestimmten Winkel.

grad: *zu drehender Winkel in Grad*

entfernen()

Entfernt den Actor von allen Funktionen der Engine. Der Actor ist nicht mehr sichtbar und wird nicht mehr gerendert.

erzeugeGelenkverbindung(**EduActor** anderer, double aX, double aY)

Baut eine drehbare Gelenkverbindung zwischen diesem und einem weiteren EduActor.

anderer: *Der andere EduActor*
aX: *x-Koordinate der Verbindung*
aY: *y-Koordinate der Verbndung*

erzeugeSeilverbindung(**EduActor** anderer, double seilLaenge, double aX, double aY, double bX, double bY)

Baut eine Seilverbindung an diesem und einem weiteren EduActor.

anderer: *Der andere AduActor*
seilLaenge: *Die Länge des Seils zwischen (aX|aY) und (bX|bY)*
aX , aY: *Der eine Seil-Endpunkt in diesem EduActor*
bX , bY: *Der andere Seil-Endpunkt in dem anderen AduActor*
Die EduActoren können sich nähern, aber einen Maximalabstand nicht überschreiten.

erzeugeStabverbindung(**EduActor** anderer, double aX, double aY, double bX, double bY)

Setzt eine Stabverbindung an diesem und einem weiteren Actor.

anderer: *Der andere AduActor*
aX , aY: *Der eine Stab-Endpunkt in diesem EduActor*
bX , bY: *Der andere Stab-Endpunkt in dem anderen AduActor*
Die EduActoren werden zu einer „Hantel“ zusammenmontiert.

getActor() : Actor *(aus der Core-Engine)*

Nicht für den Schüler-Gebrauch gedacht. Gibt das Engine-interne Actor-Objekt zurück um ggf. neue interne Features durchzureichen.

istSichtbar() : boolean

Gibt true zurück, wenn der EduAcrot gerade sichtbar ist, sonst false.

macheAktiv()

Unterwirft den EduActor von nun an den Gesetzen der Physik.

machePassiv()

Macht diesen EduActor undurchdringlich für aktive EduActoren.

macheNeutral()

Nimmt „aktiv“ bzw. „passiv“ wieder zurück.

machePartikel(doublelebenszeit)

Lässt diesen EduActor „verpuffen“

nenneDrehwinkel() :double

Gibt den aktuellen Rotationswinkel des AeduActors zurück.

nenneEbenenposition() : int

Gibt die Ebenennummer (z-Index) des Actors aus. Objekte mit höherer Nummer liegen vor Objekten mit niedrigerer Nummer.

nenneElastizitaet() : double

Gibt die aktuelle Elastizität des EduActors zurück. 1 entspricht vollkommen elastisch, 0 vollkommen inelastisch.

nenneGeschwindigkeitX() : double

Gibt die aktuelle Geschwindigkeit des EduActors in x-Richtung in m/s zurück.

nenneGeschwindigkeitY() : double

Gibt die aktuelle Geschwindigkeit des EduActors in y-Richtung in m/s zurück.

nenneMasse() : double

Gibt die aktuelle Masse des EduActors in kg zurück.

nenneMittelpunktX() : double

Gibt die aktuelle x-Koordinate des Mittelpunkts des EduActors zurück.

nenneMittelpunktY() : double

Gibt die aktuelle y-Koordinate des Mittelpunkts des EduActors zurück.

nenneReibung() : double

Gibt den aktuellen Wert des Reibungskoeffizienten zurück.

nenneTransparenz() : double

Gibt den Transparenzwert des EDU Actors aus. 0 entspricht ohne Transparenz, 1 entspricht unsichtbar.

registriereKollisionsReagierbar(ea.edu.event.KollisionsReagierbar<**EduActor**> reagierbar)

Methode zum Registrieren von Kollisionen mit irgendeinem anderen EduActor. Es muss zusätzlich die Methode boolean kollisionReagieren(EduActor ea) überschreiben werden.

reagierbar: *In der Regel eine Selbst-Referenz. Die Klasse des übergebenen Objekts muss die Methode kollisionReagieren(…) überschreiben.*

registriereKollisionsReagierbar(X anderer, ea.edu.event.KollisionsReagierbar<X> kollisionsReagierbar)

Methode zum Registrieren von Kollisionen mit einem bestimmten EduActor. Es muss zusätzlich die Methode boolean kollisionReagieren(EduActor ea) überschreiben werden.

reagierbar: *In der Regel eine Selbst-Referenz. Die Klasse des übergebenen Objekts muss die Methode kollisionReagieren(…) überschreiben.*

schneidet(EduActor objekt) : bololean

Gibt true zurück, wenn der EduActor den anderen schneidet / berührt, sonst false.

objekt: *Der andere EduActor*

setzeDrehwinkel(double grad)

Gibt den aktuellen Rotationswinkel des EduActors zurück.

grad: *zu drehender Winkel in Grad*

setzeEbenenposition(int position)

Setzt die Ebenenposition (z-Index) des EduActors. Objekte mit höheren Nummern liegen vor Objekten mit niedrigeren Nummern.

position: *Nummer der Ebene dieses EduActors*

setzeElastizitaet(double elastizitaetsKoeffizient) : double

Setzt die Elastizität dieses EduActors.

elastizitaetsKoeffizient: *1 entspricht vollkommen elastisch, 0 entspricht voll*kommen inelastisch.

setzeGeschwindigkeit(double vX, double vY)

Setze die Geschwindigkeit des EduActors auf entsprechende Werte in m/s.

vX , vY: Geschwindigkeit in x- bzw. y-Richtung in m/s.

setzeKollisionsformen(java.lang.String kollisionsFormenCode)

z.B. bei Grafiken mit transparentem Hintergrund kann man die Reaktion auf Kollisionen mit anderen Objekten der Grafik anpassen, indem man die Kollisionsform selbst definiert.

kollisionsFormenCode: *Kreis = "C mX,mY,r"*
Rechteck = "R mX,mY,hoehe,breite"
Polygon = "P aX,aY,bX,bY. ..."
Kombinationen mit &
Bspl.: "C 1,1,0.5 &R 2,3,1,2 &P 3,3,3.5,5,0.5,4,1,2"

setzeMasse(double masse)

Setzt die Masse des EduActors in kg. Dies beeinflusst z.B. das Stoßverhalten aktiver Objekte.

masse: *Die neue Masse in kg*

setzeMittelpunkt(double mX, double mY)

Setzt den Mittelpunkt des EduActors.

mX: *Die neue Mittelpunkt-x Koordinate des EduActors*
mY: *Die neue Mittelpunkt-y Koordinate des EduActors*

setzeReibung(double reibungsKoeffizient)

Setzt den Reibungskoeffizienten neu.

ReibungsKoeffizient: *Der neue Reibungkkoeffizient. 0 = reibungsfrei, 1 = starke Reibung*

setzeRotationBlockiert(boolean blockiert)

true sorgt dafür, dass der EduActor nicht rotiert, false führt z.B. nach Stößen auch zu Roration.

blockiert: *true = keine Rotation; false = Rotation erlaubt*

setzeSichtbar(boolean sichtbar)

true zeigt den EduActor an, false macht ihn unsichtbar. (Er ist aber immer noch da.)

sichtbar: *true = sichtbar; false = unsichtbar*

setzeTransparenz(double transparenz)

Setzt die Transparenz des EduActors. 0 entspricht ohne Transparenz, 1 entspricht unsichtbar.

Transparenz: *0 = ohne Transparenz; 1 = unsichtbar*

setzeWinkelgeschwindigkeit(double umdrehnungenProSekunde)

Setzt die Winkelgeschwindigkeit des EduActors.

umdrehungenProSekunde: *Anzahl der Umdrehungen pro Sekunde*

springe(double staerke)

Lässt einen aktiven EduActor senkrecht nach oben springen, wenn er gerade auf einem passiven EduActor steht.

staerke: *Stärke des Sprungs. Experimentiere mit dem Wert ...*

steht() : boolean

Gibt true zurück, wenn der EduActor gerade aktiv ist und auf einem passiven EduActor steht.

stehtAuf(**EduActor** actor) : boolean

Gibt true zurück, wenn der EduActor gerade aktiv ist und auf dem anderen passiven EduActor steht.

verschieben(double dX, double dY)

Verschiebt den EduActor in beliebige Richtung und um beliebige Entfernung.

dX: *zu verschiebende Distanz in x-Richtung*
dY: *zu verschiebende Distanz in y-Richtung*

verzoegere(double verzoegerungInSekunden, java.lang.Runnable runnable)

Führt ein „Runnable“ zeitverzögert aus. Das „Runnable“ wird am einfachsten als funktionaler Ausdruck (Lambda-Ausdruck) angegeben.

verzoegerungInSekunden: *Zeit in Sekunden, die das restliche Programm weiter läuft bis die übergebene Methode ausgeführt wird.*

Runnable: *Ein extra dafür angefertigtes Runnable-Objekt oder meist einfach ein Lambda-Ausdruck eines Methodenaufrufs. z.B.: verzoegere(2.5 , () -> setzeFarbe(„rot“))*

wirkeImpuls(double iX, double iY)

Wirkt einen Kraftstoß auf diesen EduActor aus, der zu dem übergebenen Impuls-Zugewinn (in kg*m/s) führt.

iX: *Der Impuls-Zugewinn in x-Richtung in kg*m/s*

iY: *Der Impuls-Zugewinn in y-Richtung in kg*m/s*