Scene

Params

canvas: HTMLCanvasElement
width: number (sets canvas.width)
height: number (sets canvas.height)
cssWidth: string = "100vw" (sets canvas.style.width)
cssHeight: string = "100vh" (sets canvas.style.height)

Description

The core component for Phantom 2D, used to render all components.

Methods

add(...comps): void (Adds new components to `#components`. Throws an error if any component is not a Phantom2DEntity instance.)
remove(...comps): void (removes components from `#components`)
rect(x: number, y: number, width: number, height: number, colour: string): void (draws and fills a rectangle of the colour provided)
clear(): void (clears the canvas)
img(x: number, y: number, w: number, h: number, path: string): void (draws an image onto the canvas)
render(): void (Renders each component in `#components`. Throws an error if any component is not a Phantom2DEntity instance.)
getById(id: string): Phantom2DEntity | null (Runs a find check through `#components`. Checks each component with component id equals id.)
getByAttr(name: string, value: any): Phantom2DEntity | null (Runs a find check through `#components`. Checks each component with component attribute value equals value.)
getByIdx(idx: number): Phantom2DEntity | null (returns the element in `#components` at the index of idx)
setById(id: string, newItem: Phantom2DEntity): void | null (Finds the component to be replaced by comparing component id to id, finds index of found component. Returns prematurely if the found component is undefined or the index of the found component is -1. Sets the component at the index of the found component to the new item.)
setByAttr(attr: string, value: any, newItem: Phantom2DEntity): void | null (Finds the component to be replaced by comparing component attribute value to value, finds index of found component. Returns prematurely if the found component is undefined or the index of the found component is -1. Sets the component at the index of the found component to the new item.)
setByIdx(idx: number, newItem: Phantom2DEntity): void (sets the component at index idx to the new item)
hasItem(item: Phantom2DEntity): boolean (returns a boolean of whether `#components` includes item or not)
hasItemWithId(id: string): boolean (returns a boolean of whether `#components` has an item with the id of id)
hasItemWithAttr(attr: string, value: any): boolean (returns a boolean of whether `#components` has an item with an attribute value of value)

idxOf(component: Phantom2DEntity): number (returns the index of a component within `#components`)
len(): number (returns the length of `#components`)
fillBg(colour: string): void (fills the background of the canvas)
#resolveCollisions(): void (a private function used to resolve collisions between components)
getMouseRotTo(target: Phantom2DEntity | object): number (Returns the relative rotation from the mouse to the target in radians. Requires x and y properties to work.)
getRotTo(source: Phantom2DEntity | object, target: Phantom2DEntity | object): number (Returns the relative rotation from the target to the source in radians. Requires x and y properties on both source and target to work.)
getRotToMouse(source: Phantom2DEntity | object): number (Returns the relative rotation from the source to the mouse in radians. Requires x and y properties to work.)
#isPhantom2DEntity(item: any): boolean (a private function used to check whether item is a Phantom2DEntity instance or not)
addEvent(name: string, exec: function): void (adds an event listener to the canvas)
remEvent(name: string): void (removes an event listener from the canvas)
width(): number (returns the width of the canvas)
height(): number (returns the height of the canvas)
raycast(origin: vector, angle: number, maxDist: number, filter: function): { object: Phantom2DEntity, point: { x: number, y: number }, distance: number } | null (Creates a raycast. Returns information about the hit if there is one; null if there is no hit.)
debugRay(origin: vector, angle: number, dist: number, colour: string): void (creates a visible ray)

Properties

canvas: HTMLCanvasElement
ctx: CanvasRenderingContext2D
#components: private array
mousePos: { x: number, y: number }

Phantom2DEntity

Params

expects: string[]
objname: string
settings: object

Description

The base class for all Phantom2D components. Supports custom properties through `customProperties`.

Methods

setPos(x: number, y: number): void
setRot(rad: number): void
setWidth(width: number): void
setHeight(height: number): void
getForwardVector(): { dx: number, dy: number } (returns the relative forward vector)

move(distance: number, axis: string | bit): void (Updates the position of the character. `axis` being "x" or 0 will move on the x-axis, `axis` being "y" or 1 will move on the y-axis.)
moveX(distance: number): void (updates the x position of the character by an offset of the provided distance)
moveY(distance: number): void (updates the y position of the character by an offset of the provided distance)
clampPos(min: number, max: number, axis: string | number): void (Sets the position coordinate to clamped between min and max. `axis` being "x" or 0 is x-axis, "y" or 1 is y-axis.)
clampPosX(min: number, max: number): void (sets the x position to the clamped value between min and max)
clampPosY(min: number, max: number): void (sets the y position to the clamped value between min and max)
getPos(): { x: number, y: number }
getCenter(): { x: number, y: number } (returns the center coordinate of the target)
getPosX(): number
getPosY(): number
setPosX(): void
setPosY(): void
setRanPos(min, max): void (sets position to x as a random value between `min` and `max` (inclusive) and y as a random value between `min` and `max` (inclusive))
raycast(settings: object): { object: Phantom2DEntity, point: { x: number, y: number }, distance: number } | null (Creates a raycast. Returns information about the hit if there is one; null if there is no hit.)
debugRay(settings: object): void (creates a visible ray)

Properties

id: string
shape: string
color: string
collide: function
x: number
y: number
rot: number
width: number
height: number

SceneObject (extends Phantom2DEntity)

Params

expects: string[]
objname: string
settings: object

Description

The base class for environmental objects.

Methods

none
Properties
none

StaticObject (extends SceneObject)
Params
none
Description
A basic scene shape.
Methods
update(): void (This method does nothing. It exists, because the render loop uses the `update` method on components.)
Properties
none

PhysicsObject (extends SceneObject)
Params
strength: number
Description
A shape that has physics.
Methods
update(): void (Updates physics. Increases gravity speed and y position.)
Properties
strength: number
gravspd: number (represents gravity speed)

MovingObject (extends SceneObject)
Params
speed: number
dirX: bit
dirY: bit
extentLeft: number
extentRight: number
extentDown: number
extentUp: number
Description
A shape that moves between each extent in the specified starting position. If `isBouncing` is enabled, upon reaching an extent, the shape will move in the opposite direction. `dirX` uses 0 for left direction and 1 for right direction, `dirY` uses 0 for downward direction and 1 for upward direction.
Methods
update(): void (Updates the shape's position. If an extent is reached and `isBouncing` is true, then bounces; else stops.)

Properties
speed: number
directionX: bit
directionY: bit
isBouncing: boolean
extentLeft: number
extentRight: number
extentUp: number
extentDown: number

BouncyObject (extends SceneObject)

Params
strength: number
Description
A shape that bounces colliding objects. By default, it bounces all objects. `ignore` defines objects to not bounce, `ignoreByType` defines a type of object to not bounce. `target` defines a type of object to *always* bounce, `targetByType` defines a type of object to *always* bounce.
Methods
update(): void (This method does nothing. It exists, because the render loop uses the `update` method on components.)
Properties
strength: number
ignore: array
ignoreByType: array
target: array
targetByType: array

BulletObject (extends SceneObject)

Params
clampLeft: number
clampRight: number
clampUp: number
clampDown: number
speed: number
dir: number
scene: Phantom2D.Scene
Description
An autonomously functioning shape. Acts similar to MovingObject. `dir` (0, 1, 2 or 3) represents the direction the bullet should move in. 0 represents North, 1 represents East, 2 represents South and 3 represents West. If `dir` is not 0, 1, 2 or 3, `dir` is treated as an angle value; the bullet is moved in its forward direction by the scaled vector (cos(`dir`), sin(`dir`)).
Methods

update(): void (Moves the bullet according to its direction. Removes the bullet from the scene if its x position + width is less than `clampLeft`, x position + width is more than `clampRight`, y position + height is less than `clampUp` or y position + height is more than `clampDown`.)
Properties
clampLeft: number
clampRight: number
clampUp: number
clampDown: number
speed: number
dir: number
onDestroyed: function
scene: Phantom2D.Scene

Character (extends Phantom2DEntity)
Params
expects: string[]
objname: string
settings: object
Description
The base class for character objects.
Methods
setGravSpd(newSpd: number): void (sets the components gravity speed)
jump(height): void (Sets gravity speed to the inverse of height. Entering a positive number will make it jump up, a negative number will make it jump down.)
Properties
none

PlayableCharacter (extends Character)
Params
width: number
height: number
Description
A controllable character with customizable keybinds.
Methods
update(): void (Updates gravity and y position. Executes binded actions if their associated keys are down.)
getBind(key: string): function | null (returns the associated action with a key)
setBind(key: string, action: function): void (sets a keybind to execute the provided action)
Properties
#binds: private object
#keys: private object

NonPlayableCharacter (extends Character)
Params

states: object

Description

A non-playable character with executable states.

Methods

update(): void (updates gravity and y position)
getState(name: string): function | null (returns the associated action with a state)
setState(name: string, action: function): void (sets a state to execute the provided action)
applyState(name: string): void (executes a state's action)
userInterval(name: string, delay: number): void (sets the interval to use a state's action)

Properties

#states: private object
interval: number | null