Quad tree-based collision detection engine

UniqueId:

BoundingBox:

A bounding box represents the position and size of a minimal rectangle needed to envelop a collideable instance. The position is relative to the top-left corner of its domain.

Public data members:

Note: GroupMask is a typedef for unsigned integer (width depends on the platform).

double x, y, w, h;

GroupMask groups;

Methods:

BoundingBox(double x, double y, double w, double h, GroupMask groups);

Constructs the BoundingBox object. The first 2 arguments represent the x and y coordinates of the bounding box relative to the top-left corner of its domain. The next 2 arguments are its width and height within the domain. The final argument, „groups“, is basically a bit mask. When checking for collisions, two instances can collide only if their groups overlap on at least one bit, that is:

collision = (inst1.group & inst2.group != 0);

BoundingBox();

Constructs the BoundingBox object with default values (all zeros).

bool overlaps(const BoundingBox & other) const;

Returns true if two BoundingBox objects overlap (taking their groups into consideration) and false otherwise.

bool enveloped\_by(const BoundingBox & other) const;

Checks whether a BoundingBox object is completely enveloped by another BoundingBox object. Groups are not taken into consideration here.

void reset(double x, double y, double w, double h, GroupMask groups);

With this method you can set all the fields of the BundingBox object.

Domain:

(constructor)

(double x, double y, double w, double h, size\_t maxdep, size\_t maxobj);

Constructs the domain object. The first 4 arguments specify the domain’s x and y coordinates and its width and height. Note that the domain does not use the x and y variables in any way – when you insert collideable instances their coordinates must be relative to the top-left corner of the domain. You can, however, fetch the domain’s x and y coordinates, but if you don’t need that, you can just leave them as zero. The “maxobj“ argument specifies how many objects can be in a single node of the tree before it’s split into 4 sub-nodes. The “maxdep” argument specifies the depth limit after which the nodes won’t split under any circumstances.

(destructor)

Deconstructs the domain object. Is not virtual.

void clear();

Resets the state of the domain object. After calling clear(), it will be as if it were just initially constructed.

void inst\_insert(UniqueId instance, const BoundingBox & bb);

Registers a new collideable instance with the domain object. Later, when you check for collisions, the domain will refer to this instance with the UniqueId provided here as the 1st argument. You’ll also need to refer to this instance with the same UniqueId when you want to change its position within the domain or remove it. The second argument represents the instance’s bounding box – its position, size and group within the domain.

Be careful as to not try to insert an instance whose UniqueId would overlap with another already in the domain.

void inst\_update(UniqueId instance, const BoundingBox & bb);

Updates the bounding box of an instance already present in the domain.

bool inst\_exists(UniqueId instance) const;

Checks whether an instance with the given UniqueId already exists within the domain.

void inst\_remove(UniqueId instance);

Removes an instance with the given UniqueId from the domain.

void pairs\_recalc\_start();

If the domain in question has worker threads that check for collisions, this wakes them. Otherwise, it does nothing.

size\_t pairs\_recalc\_join();

Waits for the worker threads to finish their work and return the total number of collisions found.

size\_t pairs\_recalc();

Equivalent to calling pairs\_recalc\_start(); return pairs\_recalc\_join();

bool pairs\_next(UniqueId & inst1, GenericPtr & inst2);