QUADOTS MANUAL

Srilakshmi Chintala(sc3772)
Souren Papazian(ssp2155)
Surashree Kulkarni(ssk2197)

QUADOTS MANUAL:

Functions in Control.h

avg_x

default(1) template <class elem>
float Control<elem>::avg_x() const;
custom(2) template <class elem>
float Control<elem>::avg_x(const vector<Elem_p> elements) const;

Returns the average of all x coordinates of the objects/elements present in the state for the first version and the average of the x coordinates of the vector of objects passed to the function for the second version.

Parameters: (1) none.
(2) Vector of points/objects.

Return Value: A floating point value which is the average of all x coordinates.
For (2) if the list is empty, it will return 0.

avg_y

default(1) template <class elem>
float Control<elem>::com_y() const;
custom(2) template <class elem>
float Control<elem>::com_y(const vector<Elem_p> elements) const;

Returns the average of all y coordinates of the objects/elements present in the tree for the first version and the average of the y coordinates of the vector of objects passed to the function for the second version .

Parameters: (1) none.
(2) Vector of points/objects.

Return Value: A floating point value which is the average of all y coordinates.
For (2) if the list is empty, it will return 0.

avg_dir

default(1) template <class elem>
float Control<elem>::avg_dir() const;
custom(2) template <class elem>
float Control<elem>::avg_dir(const vector<Elem_p> elements) const;

Returns the average direction of all the objects/elements present for the first version and the average direction of the vector of objects passed to the function for the second version .

Parameters: (1) none.
(2) Vector of points/objects.

Return Value: A floating point value which gives the average direction of all points.

dir_towards

```
template <class elem>
float Control<elem>::dir_towards(const Elem_p a, float x, float y) const;
```

Returns the angle/direction to move in to reach the the desired position.

Parameters: (1) Current Object/point
(2) Desired x coordinate position to move towards.
(3) Desired y coordinate position to move towards.

Return Value: A floating point value which gives the direction to move in to reach the desired goal position.

get_distance

```
template <class elem>
float Control<elem>::get_distance(const Elem_p a, const Elem_p b) const;
```

Gives the distance between two objects/points.

Parameters: (1) 1st object/point.
(2) 2nd object/point.

Return Value: A floating point value which is the distance between the 2 input objects/points .

qneighbors

```
template <class elem>
vector<shared_ptr<elem>> Control<elem>::qneighbors(shared_ptr<elem> a, float range);
```

Gives the q neighbours which fall under the given range w.r.t to the input object/point.

Parameters: (1) object/point for which we want to find the neighbours.
(2) desired range to search for neighbours.

Return Value: Returns a vector of q points/objects which lie within the range of the input point/object.

random_pos

```
template <class elem>
float Control<elem>::random_pos(int min, int max);
```

Generates a random position which falls within given a range.

Parameters: (1) minimum/lower bound of the range.
(2) maximum/upper bound of the range.

Return Value: A floating point value which is the generated random position.

Control

```
template <class elem>
Control<elem>::Control(State<elem> *s)
```

Constructor for the control.

setState

```
template <class elem>
void Control<elem>::setState(State<elem> *s)
```

Parameters: pointer to the element state.

Return Value:none.

Functions in Point.h

Point

```
Point(float x, float y, int bindex);
```

Point class constructor which sets the properties (x,y coordinates, and the behaviour index) for the point object.

Parameters: Values of x-coordinate, y-coordinate and the behaviour index.

Return Value: none.

get_id

```
int Point::get_id() const;
```

Gets the id of the point object.

Parameters: none

Return Value: Returns integer value which is the id of the point.

get_x

```
float Point::get_x() const;
```

Gets the x coordinate of the point object.

Parameters: none

Return Value: Returns float value which is the x coordinate of the point.

get_y

```
float Point::get_y() const;
```

Gets the y coordinate of the point object.

Parameters: none

Return Value: Returns float value which is the y coordinate of the point.

get_b

```
int Point::get_b() const;
```

Gets the behaviour index of the point object.

Parameters: none

Return Value: Returns integer value which is the behaviour id of the point.

set_x

```
void Point::set_x(float x);
```

Sets the x coordinate of the point object.

Parameters: value of x-coordinate

Return Value:none.

set_y

```
void Point::set_y(float y);
```

Sets the y coordinate of the point object.

Parameters: value of y-coordinate

Return Value:none.

set_b

```
void Point::set_b(int bindex);
```

Sets the behaviour index of the point object.The object will not have a behavior if the index is negative.

Parameters: the behaviour index value to be set.

Return Value: none.

update

```
void Point::update();
```

Does nothing as point objects are static.

Parameters:none.

Return Value: none.

Functions in Dot.h

Dot

```
Dot(float x, float y, float ang, float vel, unsigned int bindex);
```

Dot class constructor which sets the properties (x,y coordinates, the angle, velocity and the behaviour index) for the dot object.

Parameters: Values of x-coordinate, y-coordinate, angle, velocity and the behaviour index.

Return Value: none.

get_ang

```
float Dot::get_ang() const;
```

Gets the angle of the dot object.

Parameters: none

Return Value: Returns a floating point value which is the angle of the dot.

get_vel

```
float Dot::get_vel() const;
```

Gets the velocity of the dot object.

Parameters: none

Return Value: Returns a floating point value which is the velocity of the dot.

set_ang

```
void Dot::set_ang(float ang);
```

Sets the angle of the dot object.

Parameters: Value of the angle to be set.

Return Value:none.

set_vel

```
void Dot::set_vel(float vel);
```

Sets the velocity of the dot object.

Parameters: Value of the velocity to be set.

Return Value:none.

add_ang

```
void Dot::add_ang(float delta);
```

Adds the angle passed to the current angle of the dot object.

Parameters: Delta which is the value of the angle to be added.

Return Value:none.

add_vel

```
void Dot::add_vel(float vel);
```

Adds the velocity passed to the current velocity of the dot object.

Parameters: Vel which is the value of the velocity to be added.

Return Value:none.

update

```
void Dot::update();
```

Update location based on angle and velocity.

Parameters:none.

Return Value: none.

***Note:* all angle values are in degrees and not radians.**

Functions in Simulation.h

Simulation

```
Simulation<elem>::Simulation(double width, double height);
```

Simulation class constructor which sets the properties (width and height) for the simulation.

Parameters: Width and height.

Return Value: none.

Run

```
default(1) template <class elem>
           void Simulation<elem>::Run(int gen_count, bool print);

custom(2) template <class elem>
           void Simulation<elem>::Run(int gen_count, Renderer<elem> &renderer)
```

gen_count times (0 is infinite): For first version updates state and if print is true, print the state.For second version updates state and renders it using renderer passed in

Parameters: (1) Generate count value and bool value to specify print.
(2) Generate count value and renderer.

Return Value: none.

CreateElement

```
template <class elem>
void Simulation<elem>::CreateElement(elem e);
```

Creates a new element and returns a shared pointer to it.

Parameters: Element object to be created.

Return Value: none.

CreateBehaviour

```
template <class elem>
int Simulation<elem>::CreateBehavior(behavior &b);
```

Adds the passed behavior to the behaviors container. Returns the index it was added to give to an element.

Parameters: Behaviour to be created.

Return Value:integer value which as an index to the created behaviour.

Functions in Renderer.h

Renderer

```
Renderer(int screen_w, int screen_h, float sps);
```

Renderer class constructor which sets the properties (screen width, screen height and steps per second) for the simulation.

Parameters: Screen width, screen height and steps per second .

Return Value: none.

