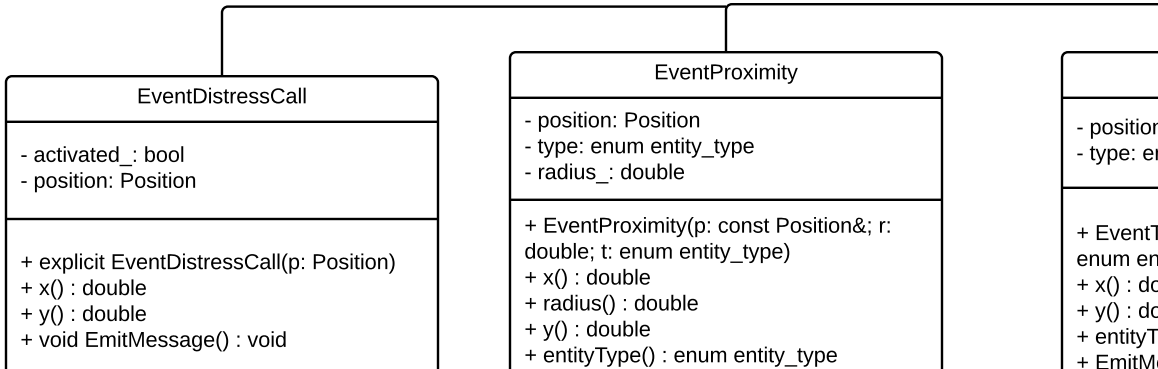
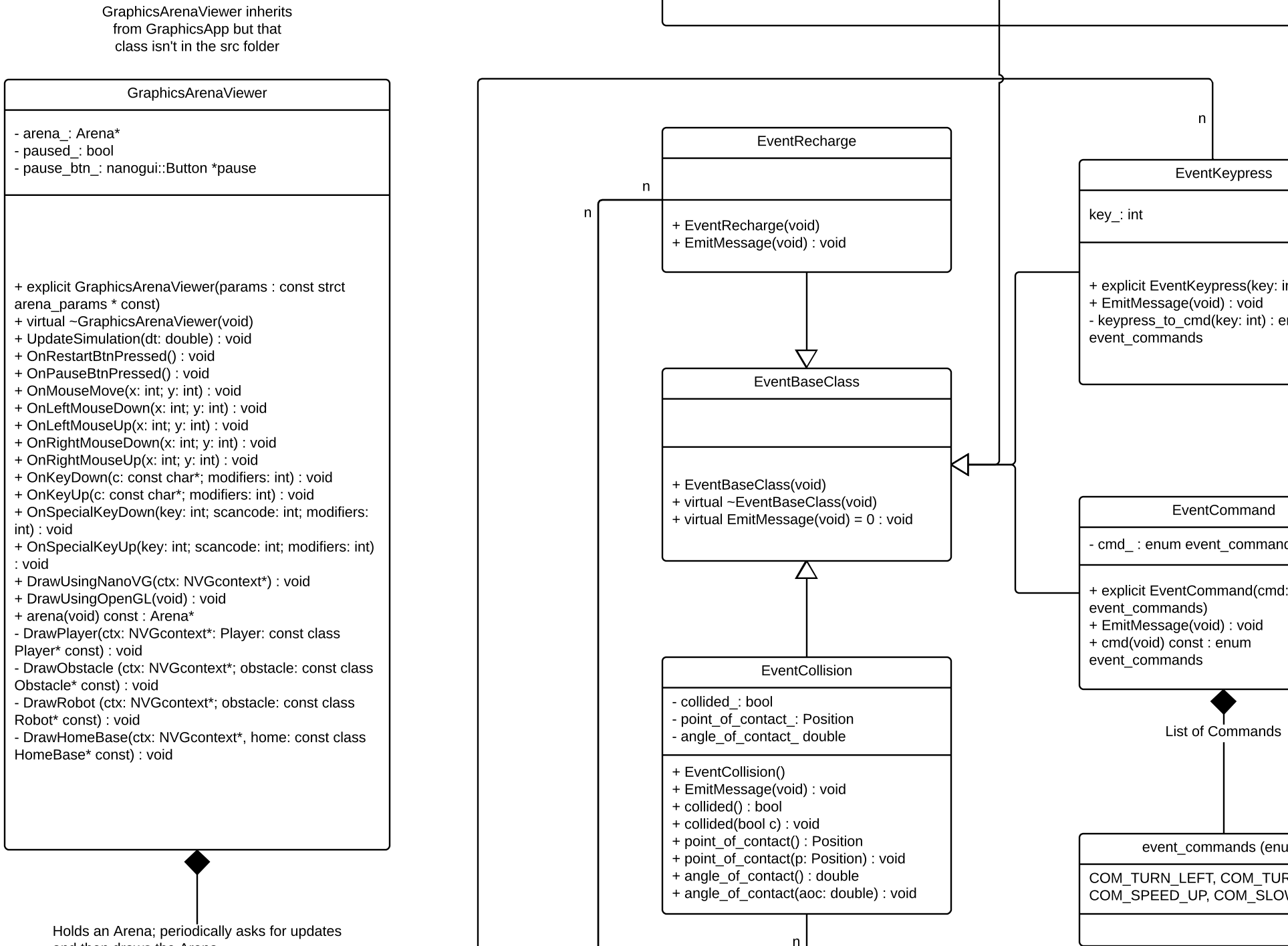
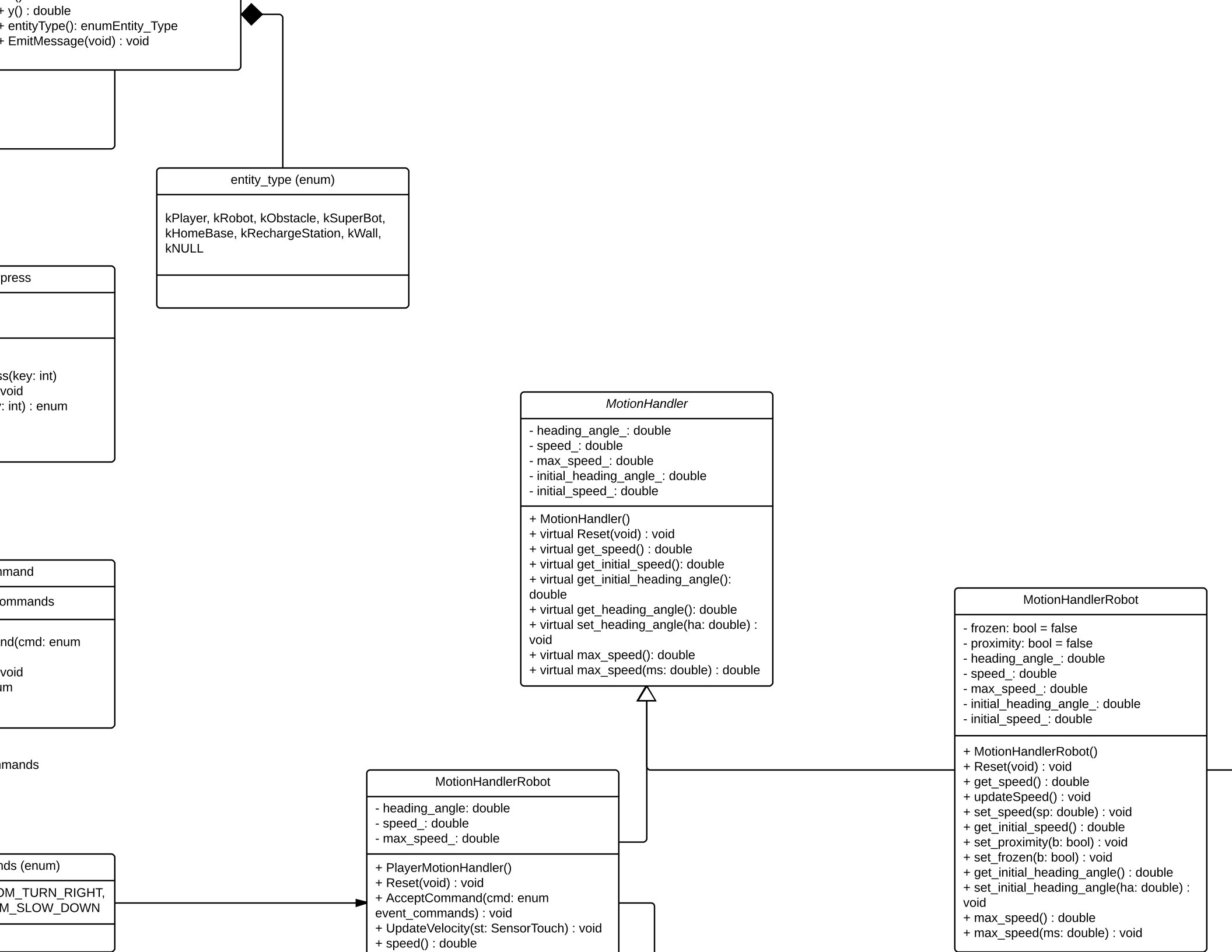
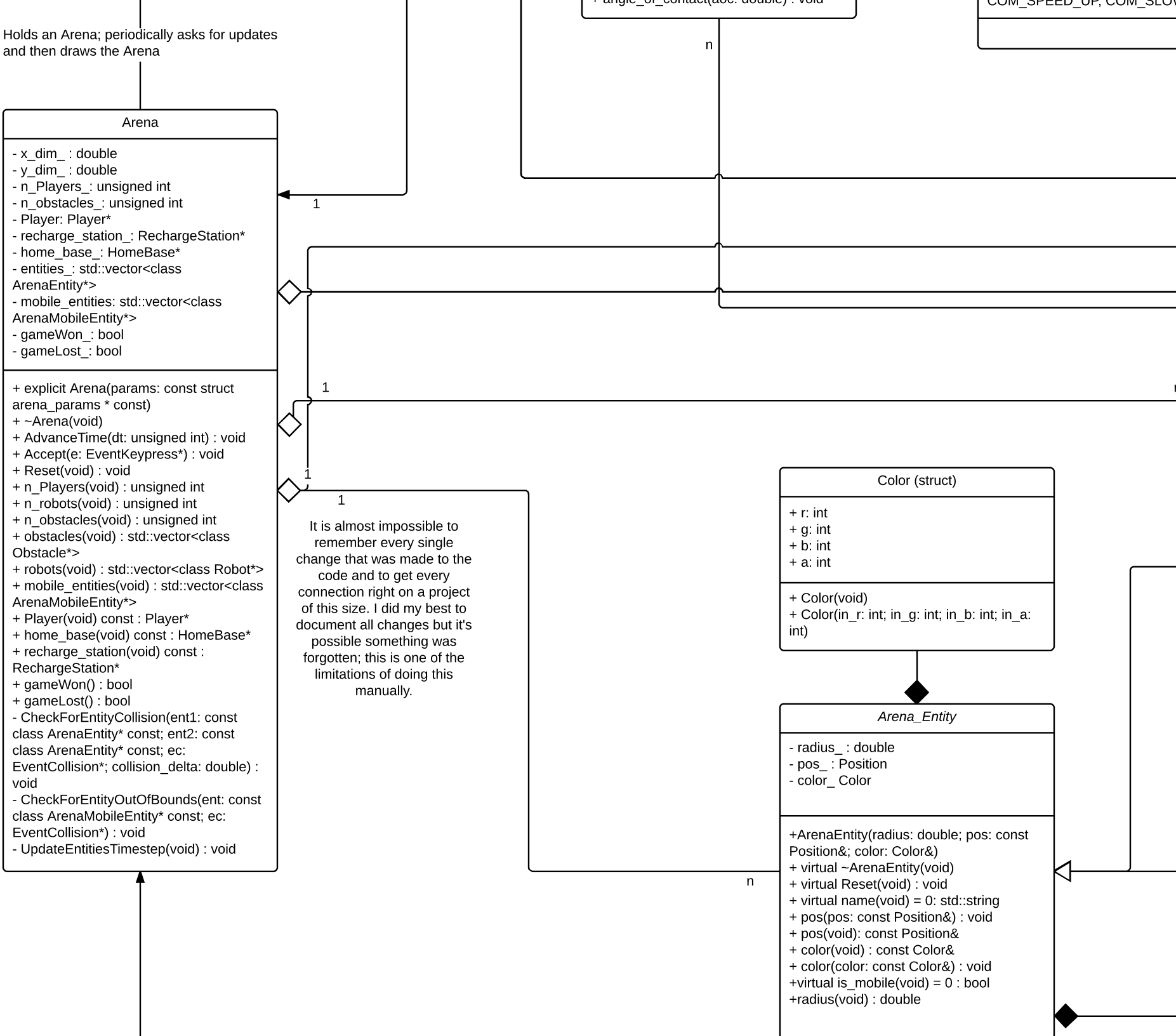
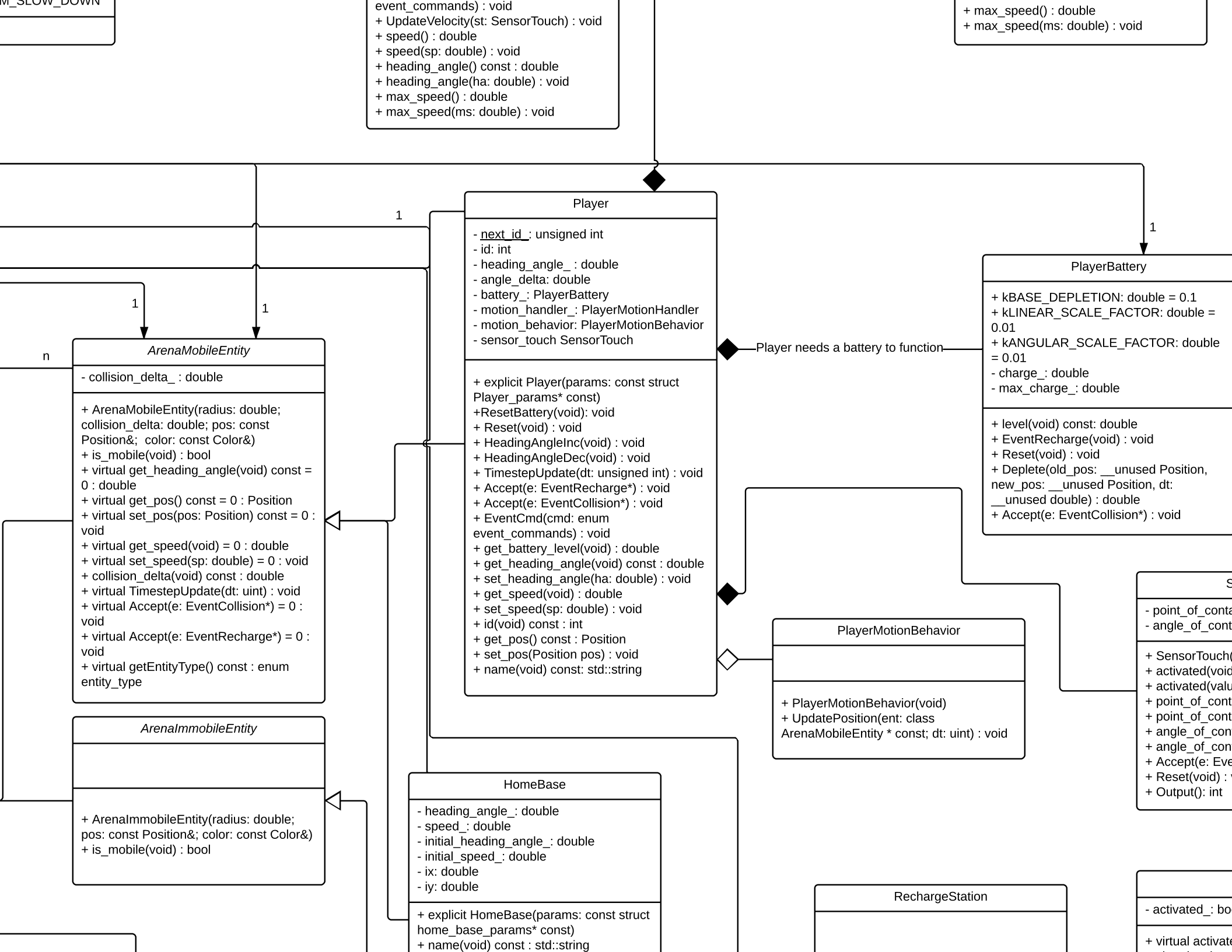

EventDistressCall
- activated_: bool
- position: Position
+ explicit EventDistressCall(p: Position)
+ x() : double
+ y() : double
+ void EmitMessage() : void
EventProximity
- position: Position
- type: enum entity_type
- radius_: double
+ EventProximity(p: const Position&; r:
double; t: enum entity_type)
+ x() : double
+ radius() : double
+ y() : double
+ entityType() : enum entity_type

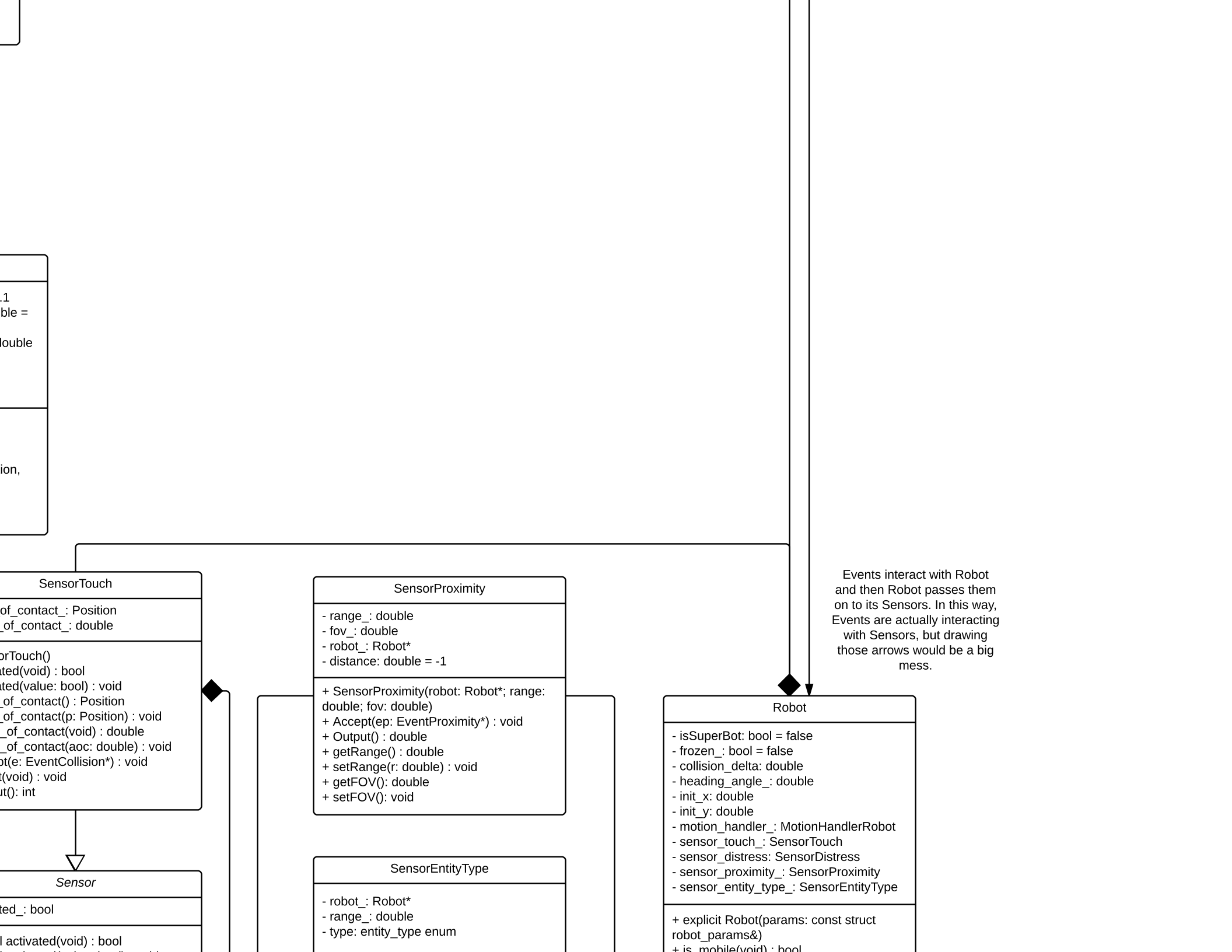
EventTypeEmit
- position Position
- type: enum entity_type
+ EventTypeEmit(p: const Position&; t:
enum entity_type)
+ x() : double
+ y() : double
+ entityType(): enumEntity_Type
+ EmitMessage(void) : void

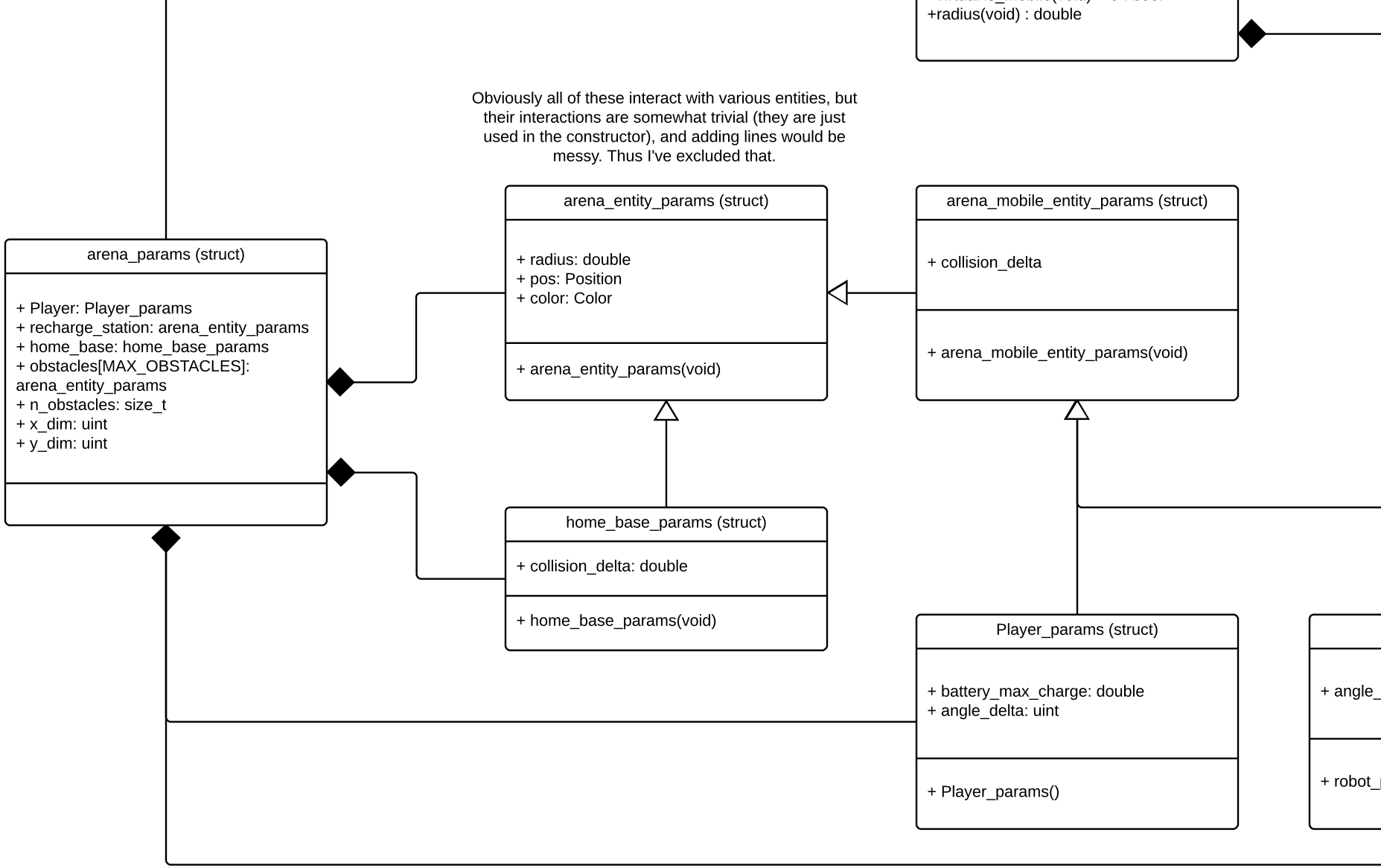

+ y() : double
+ void EmitMessage() : void
+ radius() : double
+ y() : double
+ entityType() : enum entity_type
+ EmitMessage(void) : double
GraphicsArenaViewer inherits from GraphicsApp but that class isn't in the src folder
GraphicsArenaViewer
- arena_: Arena*
- paused_: bool
- pause_btn_: nanogui::Button *pause
+ explicit GraphicsArenaViewer(params : const strct arena_params * const)
+ virtual ~GraphicsArenaViewer(void)
+ UpdateSimulation(dt: double) : void
+ OnRestartBtnPressed() : void
+ OnPauseBtnPressed() : void
+ OnMouseMove(x: int; y: int) : void
+ OnLeftMouseDown(x: int; y: int) : void
+ OnLeftMouseUp(x: int; y: int) : void
+ OnRightMouseDown(x: int; y: int) : void
+ OnRightMouseUp(x: int; y: int) : void
+ OnKeyDown(c: const char*; modifiers: int) : void
+ OnKeyUp(c: const char*; modifiers: int) : void
+ OnSpecialKeyDown(key: int; scancode: int; modifiers: int) : void
+ OnSpecialKeyUp(key: int; scancode: int; modifiers: int) : void
+ DrawUsingNanoVG(ctx: NVGcontext*) : void
+ DrawUsingOpenGL(void) : void
+ arena(void) const : Arena*
- DrawPlayer(ctx: NVGcontext*: Player: const class Player* const) : void
- DrawObstacle (ctx: NVGcontext*; obstacle: const class Obstacle* const) : void
- DrawRobot (ctx: NVGcontext*; obstacle: const class Robot* const) : void
- DrawHomeBase(ctx: NVGcontext*, home: const class HomeBase* const) : void
Holds an Arena; periodically asks for updates
EventRecharge
+ EventRecharge(void)
+ EmitMessage(void) : void
EventKeypress
key_: int
+ explicit EventKeypress(key: i
+ EmitMessage(void) : void
- keypress_to_cmd(key: int) : e
event_commands
EventBaseClass
+ EventBaseClass(void)
+ virtual ~EventBaseClass(void)
+ virtual EmitMessage(void) = 0 : void
EventCommand
- cmd_ : enum event_comman
+ explicit EventCommand(cmd:
event_commands)
+ EmitMessage(void) : void
+ cmd(void) const : enum
event_commands
List of Commands
EventCollision
- collided_: bool
- point_of_contact_: Position
- angle_of_contact_ double
+ EventCollision()
+ EmitMessage(void) : void
+ collided() : bool
+ collided(bool c) : void
+ point_of_contact() : Position
+ point_of_contact(p: Position) : void
+ angle_of_contact() : double
+ angle_of_contact(aoc: double) : void
event_commands (enu
COM_TURN_LEFT, COM_TUR
COM_SPEED_UP, COM_SLO
n
n
n
n

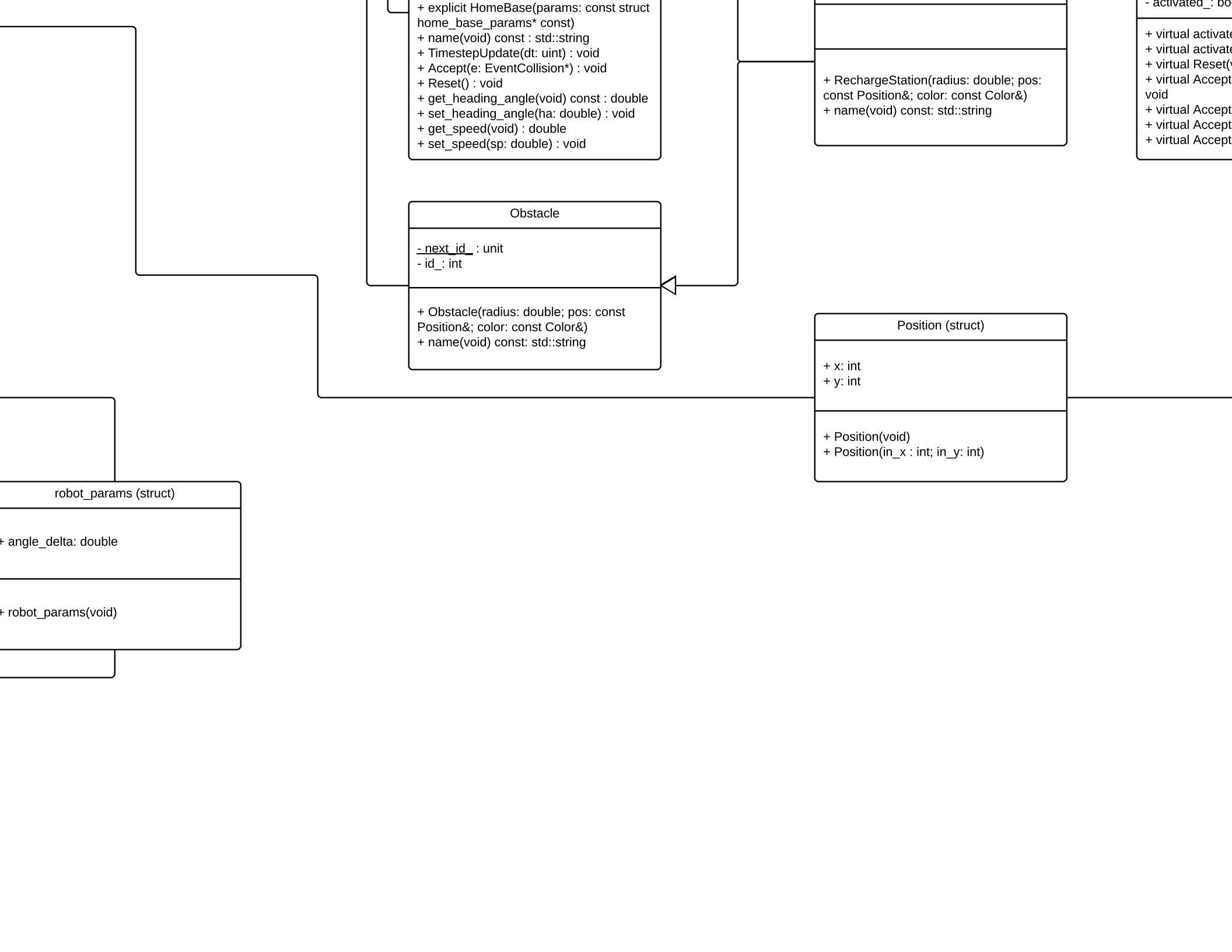

+ y() : double
+ entityType(): enumEntity_Type
+ EmitMessage(void) : void
entity_type (enum)
kPlayer, kRobot, kObstacle, kSuperBot, kHomeBase, kRechargeStation, kWall, kNULL
press
ss(key: int)
void
: int) : enum
mmand
ommands
nd(cmd: enum
void
um
mmands
nds (enum)
OM_TURN_RIGHT, M_SLOW_DOWN
MotionHandler
- heading_angle_: double
- speed_: double
- max_speed_: double
- initial_heading_angle_: double
- initial_speed_: double
+ MotionHandler()
+ virtual Reset(void) : void
+ virtual get_speed() : double
+ virtual get_initial_speed(): double
+ virtual get_initial_heading_angle(): double
+ virtual get_heading_angle(): double
+ virtual set_heading_angle(ha: double) : void
+ virtual max_speed(): double
+ virtual max_speed(ms: double) : double
MotionHandlerRobot
- heading_angle: double
- speed_: double
- max_speed_: double
+ PlayerMotionHandler()
+ Reset(void) : void
+ AcceptCommand(cmd: enum event_commands) : void
+ UpdateVelocity(st: SensorTouch) : void
+ speed() : double
MotionHandlerRobot
- frozen: bool = false
- proximity: bool = false
- heading_angle_: double
- speed_: double
- max_speed_: double
- initial_heading_angle_: double
- initial_speed_: double
+ MotionHandlerRobot()
+ Reset(void) : void
+ get_speed() : double
+ updateSpeed() : void
+ set_speed(sp: double) : void
+ get_initial_speed() : double
+ set_proximity(b: bool) : void
+ set_frozen(b: bool) : void
+ get_initial_heading_angle() : double
+ set_initial_heading_angle(ha: double) : void
+ max_speed() : double
+ max_speed(ms: double) : void

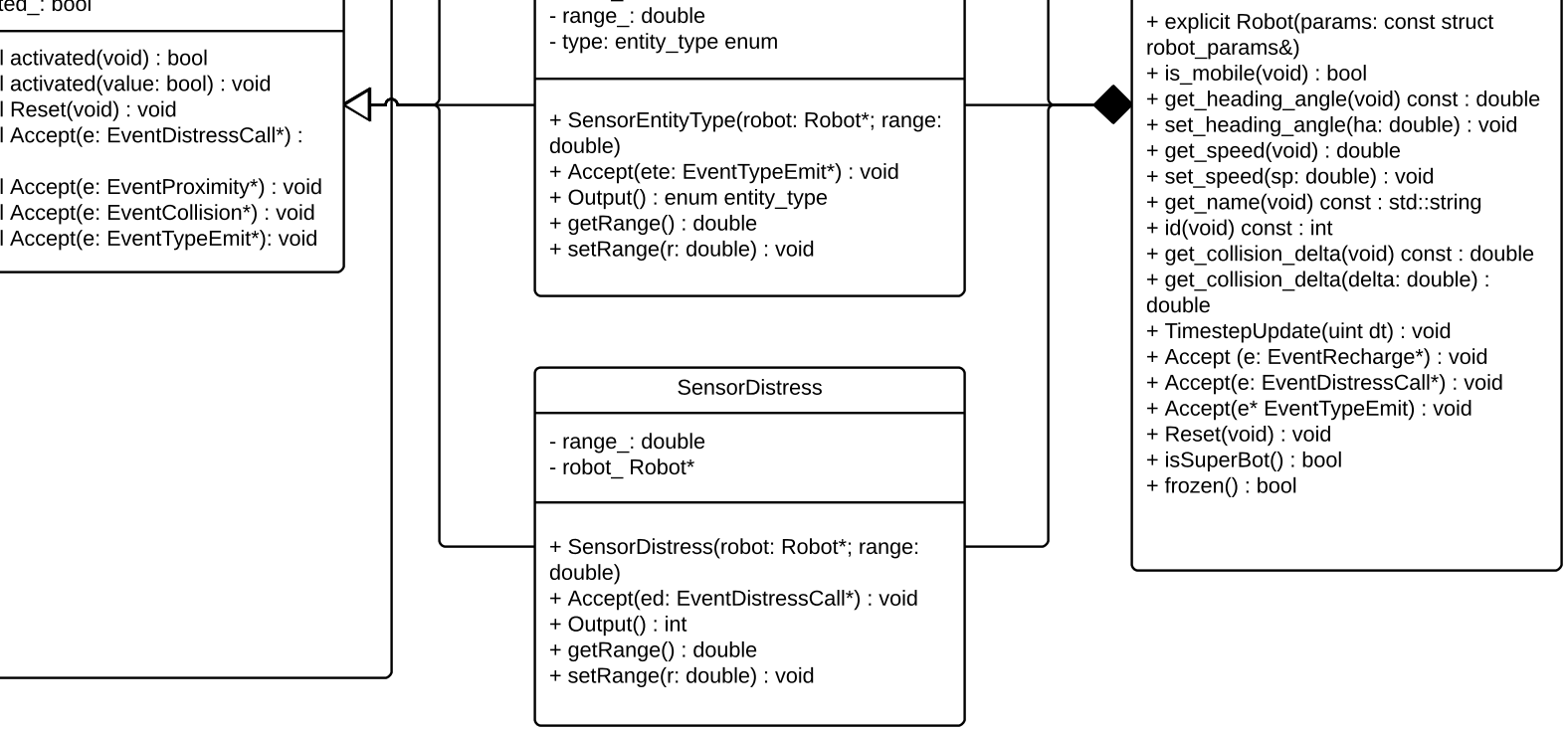

Holds an Arena; periodically asks for updates and then draws the Arena
Arena
- x_dim_ : double
- y_dim_ : double
- n_Players_: unsigned int
- n_obstacles_: unsigned int
- Player: Player*
- recharge_station_: RechargeStation*
- home_base_: HomeBase*
- entities_: std::vector<class ArenaEntity*>
- mobile_entities: std::vector<class ArenaMobileEntity*>
- gameWon_: bool
- gameLost_: bool
+ explicit Arena(params: const struct arena_params * const)
+ ~Arena(void)
+ AdvanceTime(dt: unsigned int) : void
+ Accept(e: EventKeypress*) : void
+ Reset(void) : void
+ n_Players(void) : unsigned int
+ n_robots(void) : unsigned int
+ n_obstacles(void) : unsigned int
+ obstacles(void) : std::vector<class Obstacle*>
+ robots(void) : std::vector<class Robot*>
+ mobile_entities(void) : std::vector<class ArenaMobileEntity*>
+ Player(void) const : Player*
+ home_base(void) const : HomeBase*
+ recharge_station(void) const : RechargeStation*
+ gameWon() : bool
+ gameLost() : bool
- CheckForEntityCollision(ent1: const class ArenaEntity* const; ent2: const class ArenaEntity* const; ec: EventCollision*; collision_delta: double) : void
- CheckForEntityOutOfBounds(ent: const class ArenaMobileEntity* const; ec: EventCollision*) : void
- UpdateEntitiesTimestep(void) : void
1
n
1
1
1
It is almost impossible to remember every single change that was made to the code and to get every connection right on a project of this size. I did my best to document all changes but it's possible something was forgotten; this is one of the limitations of doing this manually.
Color (struct)
+ r: int
+ g: int
+ b: int
+ a: int
+ Color(void)
+ Color(in_r: int; in_g: int; in_b: int; in_a: int)
n
Arena_Entity
- radius_ : double
- pos_ : Position
- color_ Color
+ ArenaEntity(radius: double; pos: const Position&; color: Color&)
+ virtual ~ArenaEntity(void)
+ virtual Reset(void) : void
+ virtual name(void) = 0: std::string
+ pos(pos: const Position&) : void
+ pos(void): const Position&
+ color(void) : const Color&
+ color(color: const Color&) : void
+virtual is_mobile(void) = 0 : bool
+radius(void) : double


M_SLOW_DOWN
event_commands) : void
+ UpdateVelocity(st: SensorTouch) : void
+ speed() : double
+ speed(sp: double) : void
+ heading_angle() const : double
+ heading_angle(ha: double) : void
+ max_speed() : double
+ max_speed(ms: double) : void
+ max_speed() : double
+ max_speed(ms: double) : void
1
1
1
1
n
Player
- next_id_: unsigned int
- id: int
- heading_angle_ : double
- angle_delta: double
- battery_: PlayerBattery
- motion_handler_: PlayerMotionHandler
- motion_behavior: PlayerMotionBehavior
- sensor_touch SensorTouch
+ explicit Player(params: const struct Player_params* const)
+ResetBattery(void): void
+ Reset(void) : void
+ HeadingAngleInc(void) : void
+ HeadingAngleDec(void) : void
+ TimestepUpdate(dt: unsigned int) : void
+ Accept(e: EventRecharge*) : void
+ Accept(e: EventCollision*) : void
+ EventCmd(cmd: enum event_commands) : void
+ get_battery_level(void) : double
+ get_heading_angle(void) const : double
+ set_heading_angle(ha: double) : void
+ get_speed(void) : double
+ set_speed(sp: double) : void
+ id(void) const : int
+ get_pos() const : Position
+ set_pos(Position pos) : void
+ name(void) const: std::string
Player needs a battery to function
PlayerBattery
+ kBASE_DEPLETION: double = 0.1
+ kLINEAR_SCALE_FACTOR: double = 0.01
+ kANGULAR_SCALE_FACTOR: double = 0.01
- charge_: double
- max_charge_: double
+ level(void) const: double
+ EventRecharge(void) : void
+ Reset(void) : void
+ Deplete(old_pos: __unused Position, new_pos: __unused Position, dt: __unused double) : double
+ Accept(e: EventCollision*) : void
ArenaMobileEntity
- collision_delta_ : double
+ ArenaMobileEntity(radius: double; collision_delta: double; pos: const Position&; color: const Color&)
+ is_mobile(void) : bool
+ virtual get_heading_angle(void) const = 0 : double
+ virtual get_pos() const = 0 : Position
+ virtual set_pos(pos: Position) const = 0 : void
+ virtual get_speed(void) = 0 : double
+ virtual set_speed(sp: double) = 0 : void
+ collision_delta(void) const : double
+ virtual TimestepUpdate(dt: uint) : void
+ virtual Accept(e: EventCollision*) = 0 : void
+ virtual Accept(e: EventRecharge*) = 0 : void
+ virtual getEntityType() const : enum entity_type
PlayerMotionBehavior
+ PlayerMotionBehavior(void)
+ UpdatePosition(ent: class ArenaMobileEntity * const; dt: uint) : void
ArenaImmobileEntity
+ ArenaImmobileEntity(radius: double; pos: const Position&; color: const Color&)
+ is_mobile(void) : bool
HomeBase
- heading_angle_: double
- speed_: double
- initial_heading_angle_: double
- initial_speed_: double
- ix: double
- iy: double
+ explicit HomeBase(params: const struct home_base_params* const)
+ name(void) const : std::string
RechargeStation
- point_of_conta
- angle_of_cont
+ SensorTouch(
+ activated(void
+ activated(valu
+ point_of_cont
+ point_of_cont
+ angle_of_con
+ angle_of_con
+ Accept(e: Eve
+ Reset(void) :
+ Output(): int
- activated_: bo
+ virtual activate


.1
ble =
louble
ion,
SensorTouch
of_contact_: Position
_of_contact_: double
orTouch()
ated(void) : bool
ated(value: bool) : void
_of_contact() : Position
_of_contact(p: Position) : void
_of_contact(void) : double
_of_contact(aoc: double) : void
ot(e: EventCollision*) : void
t(void) : void
ut(): int
Sensor
ted_: bool
l activated(void) : bool
SensorProximity
- range_: double
- fov_: double
- robot_: Robot*
- distance: double = -1
+ SensorProximity(robot: Robot*; range: double; fov: double)
+ Accept(ep: EventProximity*) : void
+ Output() : double
+ getRange() : double
+ setRange(r: double) : void
+ getFOV(): double
+ setFOV(): void
SensorEntityType
- robot_: Robot*
- range_: double
- type: entity_type enum
Robot
- isSuperBot: bool = false
- frozen_: bool = false
- collision_delta: double
- heading_angle_: double
- init_x: double
- init_y: double
- motion_handler_: MotionHandlerRobot
- sensor_touch_: SensorTouch
- sensor_distress: SensorDistress
- sensor_proximity_: SensorProximity
- sensor_entity_type_: SensorEntityType
+ explicit Robot(params: const struct robot_params&)
+ is_mobile(void) : bool
Events interact with Robot and then Robot passes them on to its Sensors. In this way, Events are actually interacting with Sensors, but drawing those arrows would be a big mess.


+radius(void) : double
Obviously all of these interact with various entities, but their interactions are somewhat trivial (they are just used in the constructor), and adding lines would be messy. Thus I've excluded that.
arena_entity_params (struct)
+ radius: double
+ pos: Position
+ color: Color
+ arena_entity_params(void)
arena_mobile_entity_params (struct)
+ collision_delta
+ arena_mobile_entity_params(void)
arena_params (struct)
+ Player: Player_params
+ recharge_station: arena_entity_params
+ home_base: home_base_params
+ obstacles[MAX_OBSTACLES]: arena_entity_params
+ n_obstacles: size_t
+ x_dim: uint
+ y_dim: uint
home_base_params (struct)
+ collision_delta: double
+ home_base_params(void)
Player_params (struct)
+ battery_max_charge: double
+ angle_delta: uint
+ Player_params()
+ angle_
+ robot_


+ explicit HomeBase(params: const struct home_base_params* const)
+ name(void) const : std::string
+ TimestepUpdate(dt: uint) : void
+ Accept(e: EventCollision*) : void
+ Reset() : void
+ get_heading_angle(void) const : double
+ set_heading_angle(ha: double) : void
+ get_speed(void) : double
+ set_speed(sp: double) : void
+ RechargeStation(radius: double; pos: const Position&; color: const Color&)
+ name(void) const: std::string
- activated_: bo
+ virtual activate
+ virtual activate
+ virtual Reset(v
+ virtual Accept
void
+ virtual Accept
+ virtual Accept
+ virtual Accept
Obstacle
- next_id_ : unit
- id_: int
+ Obstacle(radius: double; pos: const Position&; color: const Color&)
+ name(void) const: std::string
Position (struct)
+ x: int
+ y: int
+ Position(void)
+ Position(in_x : int; in_y: int)
robot_params (struct)
+ angle_delta: double
+ robot_params(void)


ted_: bool
l activated(void) : bool
l activated(value: bool) : void
l Reset(void) : void
l Accept(e: EventDistressCall*) :
l Accept(e: EventProximity*) : void
l Accept(e: EventCollision*) : void
l Accept(e: EventTypeEmit*): void
- range_: double
- type: entity_type enum
+ SensorEntityType(robot: Robot*; range: double)
+ Accept(ete: EventTypeEmit*) : void
+ Output() : enum entity_type
+ getRange() : double
+ setRange(r: double) : void
SensorDistress
- range_: double
- robot_ Robot*
+ SensorDistress(robot: Robot*; range: double)
+ Accept(ed: EventDistressCall*) : void
+ Output() : int
+ getRange() : double
+ setRange(r: double) : void
+ explicit Robot(params: const struct robot_params&)
+ is_mobile(void) : bool
+ get_heading_angle(void) const : double
+ set_heading_angle(ha: double) : void
+ get_speed(void) : double
+ set_speed(sp: double) : void
+ get_name(void) const : std::string
+ id(void) const : int
+ get_collision_delta(void) const : double
+ get_collision_delta(delta: double) : double
+ TimestepUpdate(uint dt) : void
+ Accept (e: EventRecharge*) : void
+ Accept(e: EventDistressCall*) : void
+ Accept(e* EventTypeEmit) : void
+ Reset(void) : void
+ isSuperBot() : bool
+ frozen() : bool