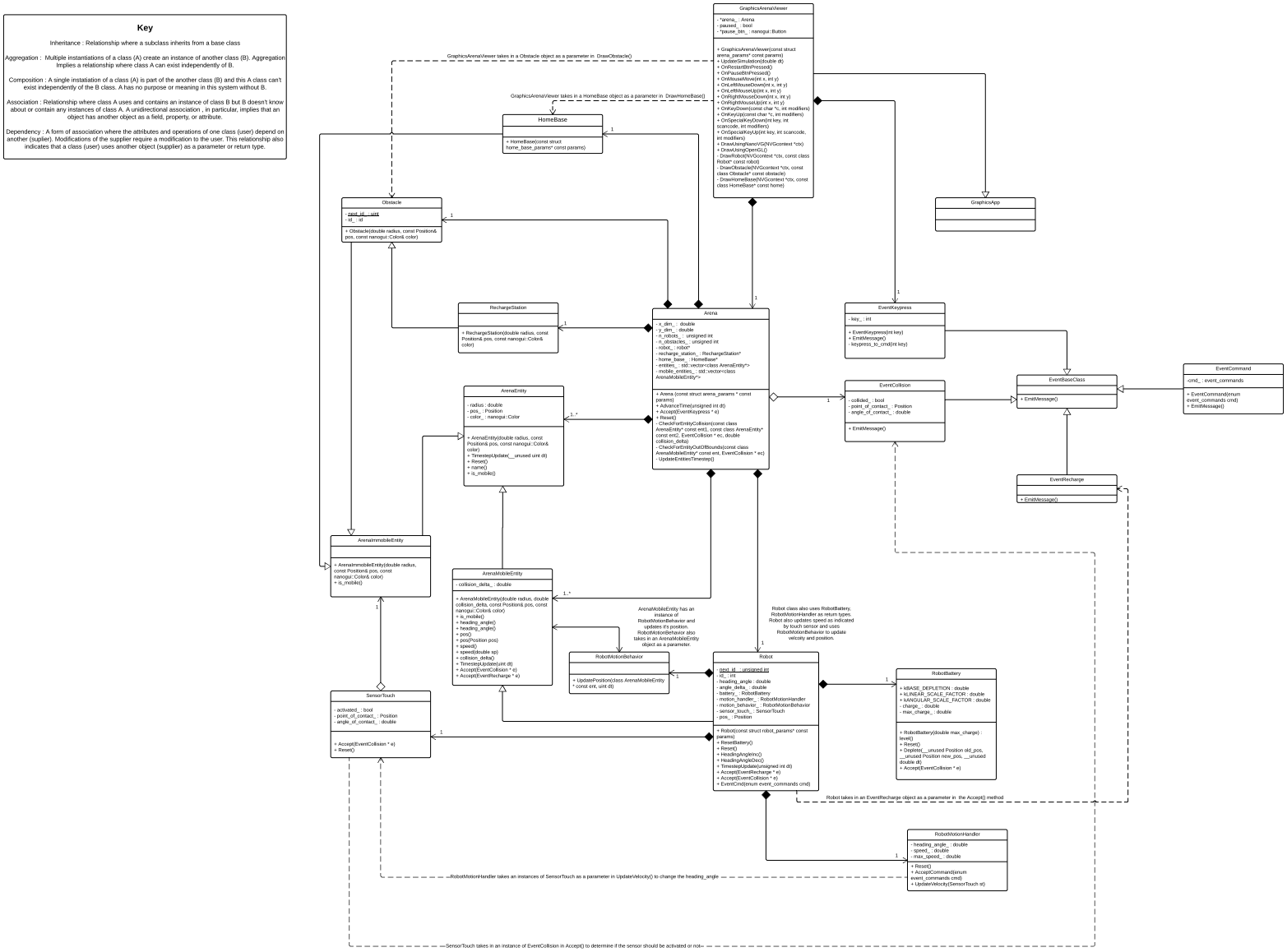
Key
Inheritance : Relationship where a subclass inherits from a base class
Aggregation : Multiple instantiations of a class (A) create an instance of another class (B). Aggregation Implies a relationship where class A can exist independently of B.
Composition : A single instatiation of a class (A) is part of the another class (B) and this A class can't exist independently of the B class. A has no purpose or meaning in this system without B.
Association : Relationship where class A uses and contains an instance of class B but B doesn't know about or contain any instances of class A. A unidirectional association , in particular, implies that an object has another object as a field, property, or attribute.
Dependency : A form of association where the attributes and operations of one class (user) depend on another (supplier). Modifications of the supplier require a modification to the user. This relationship also indicates that a class (user) uses another object (supplier) as a parameter or return type.
GraphicsArenaViewer
- *arena_ : Arena
- paused_ : bool
- *pause_btn_ : nanogui::Button
+ GraphicsArena/Viewer(const struct arena_params* const params)
+ UpdateSimulation(double dt)
+ OnRestartBtnPressed()
+ OnPausedBtnPressed()
+ OnMouseMove(int x, int y)
+ OnLeftMouseDown(int x, int y)
+ OnLeftMouseUp(int x, int y)
+ OnRightMouseDown(int x, int y)
+ OnRightMouseUp(int x, int y)
+ OnKeyDown(const char *c, int modifiers)
+ OnKeyUp(const char *c, int modifiers)
+ OnSpecialKeyDown(int key, int scancode, int modifiers)
+ OnSpecialKeyUp(int key, int scancode, int modifiers)
+ DrawUsingNanoVG(NVGcontext *ctx)
+ DrawUsingOpenGL()
- DrawRobot(NVGcontext *ctx, const class Robot* const robot)
- DrawObstacle(NVGcontext *ctx, const class Obstacle* const obstacle)
- DrawHomeBase(NVGcontext *ctx, const class HomeBase* const home)
GraphicsArenaViewer takes in a Obstacle object as a parameter in DrawObstacle()
GraphicsArenaViewer takes in a HomeBase object as a parameter in DrawHomeBase()
HomeBase
+ HomeBase(const struct home_base_params* const params)
GraphicsApp
Obstacle
- next_id_ : uint
- id_ : int
+ Obstacle(double radius, const Position& pos, const nanogui::Color& color)
RechargeStation
+ RechargeStation(double radius, const Position& pos, const nanogui::Color& color)
Arena
- x_dim_ : double
- y_dim_ : double
- n_robots_ : unsigned int
- n_obstacles_ : unsigned int
- robot_ : robot*
- recharge_station_ : RechargeStation*
- home_base_ : HomeBase*
- entities_ : std::vector<class ArenaEntity*>
- mobile_entities_ : std::vector<class ArenaMobileEntity*>
+ Arena (const struct arena_params * const params)
+ AdvanceTime(unsigned int dt)
+ Accept(EventKeypress * e)
+ Reset()
- CheckForEntityCollision(const class ArenaEntity* const ent1, const class ArenaEntity* const ent2, EventCollision * ec, double collision_delta)
- CheckForEntityOutOfBounds(const class ArenaMobileEntity* const ent, EventCollision * ec)
- UpdateEntitiesTimestep()
EventKeypress
- key_ : int
+ EventKeypress(int key)
+ EmitMessage()
- keypress_to_cmd(int key)
EventCollision
- collided_ : bool
- point_of_contact_ : Position
- angle_of_contact_ : double
+ EmitMessage()
EventBaseClass
+ EmitMessage()
EventCommand
-cmd_ : event_commands
+ EventCommand(enum event_commands cmd)
+ EmitMessage()
EventRecharge
+ EmitMessage()
ArenaEntity
- radius : double
- pos_ : Position
- color_ : nanogui::Color
+ ArenaEntity(double radius, const Position& pos, const nanogui::Color& color)
+ TimestepUpdate(__unused unit dt)
+ Reset()
+ name()
+ is_mobile()
ArenaImmobileEntity
+ ArenaImmobileEntity(double radius, const Position& pos, const nanogui::Color& color)
+ is_mobile()
ArenaMobileEntity
- collision_delta_ : double
+ ArenaMobileEntity(double radius, double collision_delta, const Position& pos, const nanogui::Color& color)
+ is_mobile()
+ heading_angle()
+ heading_angle()
+ pos()
+ pos(Position pos)
+ speed()
+ speed(double sp)
+ collision_delta()
+ TimestepUpdate(uint dt)
+ Accept(EventCollision * e)
+ Accept(EventRecharge * e)
ArenaMobileEntity has an instance of RobotMotionBehavior and updates it's position. RobotMotionBehavior also takes in an ArenaMobileEntity object as a parameter.
RobotMotionBehavior
+ UpdatePosition(class ArenaMobileEntity * const ent, unit dt)
Robot
- next_id_ : unsigned int
- id_ : int
- heading_angle : double
- angle_delta_ : double
- battery_ : RobotBattery
- motion_handler_ : RobotMotionHandler
- motion_behavior_ : RobotMotionBehavior
- sensor_touch_ : SensorTouch
- pos_ : Position
+ Robot(const struct robot_params* const params)
+ ResetBattery()
+ Reset()
+ HeadingAngleInc()
+ HeadingAngleDec()
+ TimestepUpdate(unsigned int dt)
+ Accept(EventRecharge * e)
+ Accept(EventCollision * e)
+ EventCmd(enum event_commands cmd)
Robot class also uses RobotBattery, RobotMotionHandler as return types. Robot also updates speed as indicated by touch sensor and uses RobotMotionBehavior to update velocity and position.
RobotBattery
+ kBASE_DEPLETION : double
+ kLINEAR_SCALE_FACTOR : double
+ kANGULAR_SCALE_FACTOR : double
- charge_ : double
- max_charge_ : double
+ RobotBattery(double max_charge) : level()
+ Reset()
+ Deplete(__unused Position old_pos, __unused Position new_pos, __unused double dt)
+ Accept(EventCollision * e)
SensorTouch
- activated_ : bool
- point_of_contact_ : Position
- angle_of_contact_ : double
+ Accept(EventCollision * e)
+ Reset()
Robot takes in an EventRecharge object as a parameter in the Accept() method
RobotMotionHandler
- heading_angle_ : double
- speed_ : double
- max_speed_ : double
+ Reset()
+ AcceptCommand(enum event_commands cmd)
+ UpdateVelocity(SensorTouch st)
RobotMotionHandler takes an instances of SensorTouch as a parameter in UpdateVelocity() to change the heading_angle
SensorTouch takes in an instance of EventCollision in Accept() to determine if the sensor should be activated or not

UML Write-Up

The purpose of the software is to simulate robots (Robot class) in an Arena (Arena class). The three main parts of the software are the Arena, Robot and GraphicsArenaViewer classes that all interact with interact with event classes that inherit from the EventBaseClass. The robots will be displayed through the GraphicsArenaViewer class that controls the Arena class, tracks key and mouse events (EventKeyPress), and perform the actual drawings like HomeBase on the graphics window. The Arena class contains important properties like the robot recharge station (RechargeStation), obstacles (Obstacle), and arena entities (ArenaEntity / ArenaMobileEntity / ArenaImmobileEntity) that provide vital information like speed and position of the entities (i.e. robots) in the arena. The Robot class contains different properties like the battery (RobotBattery), sensor (SensorTouch), and motion handling/behavior (RobotMotionHandler and RobotMotionBehavior) of the robots themselves that maintain vital information like the speed, heading angle, and points of contact with other arena objects. Lastly, both Robot and Arena associate themselves with events that all inherit from EventBaseClass. Specifically, Arena uses EventCollision to track collisions while Robot consults EventRecharge to output a message whenever the robot battery has been recharged.