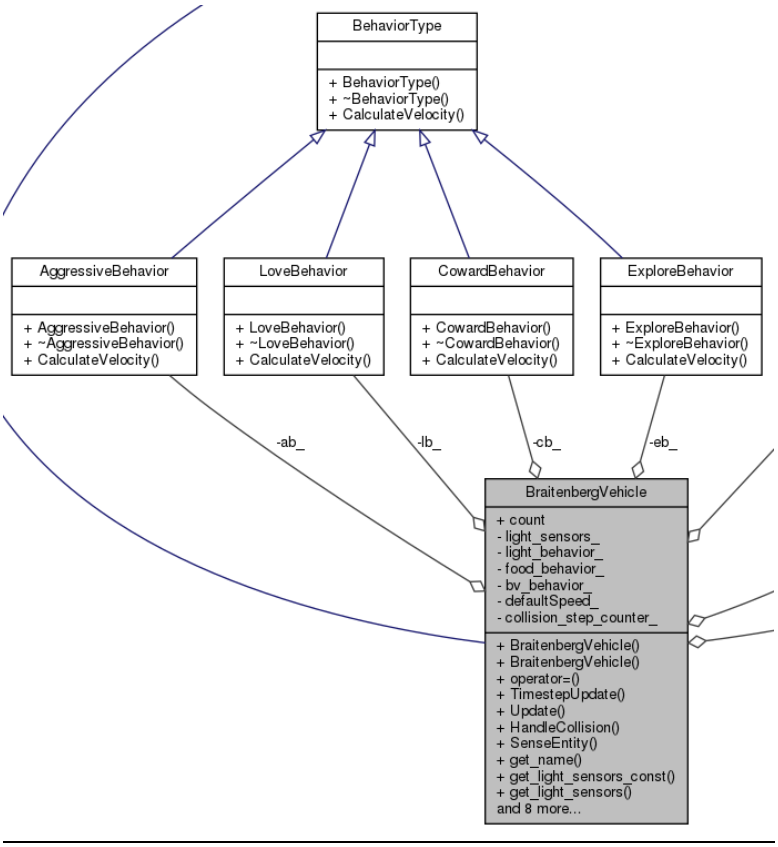
Moti Begna

CSCI 3081

4/1/2019

Iteration 2 Preliminary 1

Doxy Generated UML for Strategy Pattern



BV method using Strategy/Behavior

```
void BraitenbergVehicle::Update() {
  WheelVelocity light_wheel_velocity = WheelVelocity(0, 0);
  int numBehaviors = 3;

  switch (light_behavior_) {
    case kExplore:
      light_wheel_velocity = eb_.CalculateVelocity(
        closest_light_entity_, defaultSpeed_, light_sensors_);
      break;
```

```
    case kAggressive:
      light_wheel_velocity = ab_.CalculateVelocity(
        closest_light_entity_, defaultSpeed_, light_sensors_);
      break;
    case kLove:
      light_wheel_velocity = lb_.CalculateVelocity(
        closest_light_entity_, defaultSpeed_, light_sensors_);
      break;
    case kCoward:
      light_wheel_velocity = cb_.CalculateVelocity(
        closest_light_entity_, defaultSpeed_, light_sensors_);
      break;
    case kNone:
    default:
      numBehaviors--;
      break;
  }

  WheelVelocity food_wheel_velocity = WheelVelocity(0, 0);

  switch (food_behavior_) {
    case kExplore:
      food_wheel_velocity = eb_.CalculateVelocity(
        closest_food_entity_, defaultSpeed_, light_sensors_);
      break;
    case kAggressive:
      food_wheel_velocity = ab_.CalculateVelocity(
        closest_food_entity_, defaultSpeed_, light_sensors_);
      break;
    case kLove:
      food_wheel_velocity = lb_.CalculateVelocity(
        closest_food_entity_, defaultSpeed_, light_sensors_);
      break;
    case kCoward:
      food_wheel_velocity = cb_.CalculateVelocity(
        closest_food_entity_, defaultSpeed_, light_sensors_);
      break;
    case kNone:
    default:
      numBehaviors--;
      break;
  }
...
```

BV method using BV “sensor”

```
void BraitenbergVehicle::SenseEntity(const ArenaEntity& entity) {
  const ArenaEntity** closest_entity_ = NULL;
  if (entity.get_type() == kLight) {
    closest_entity_ = &closest_light_entity_;
  } else if (entity.get_type() == kFood) {
    closest_entity_ = &closest_food_entity_;
  } else if (entity.get_type() == kBraitenberg) {
    closest_entity_ = &closest_bv_entity_;
  }
…

void BraitenbergVehicle::Update() {
…
WheelVelocity bv_wheel_velocity = WheelVelocity(0, 0);

  switch (bv_behavior_) {
    case kExplore:
      bv_wheel_velocity = eb_.CalculateVelocity(
        closest_bv_entity_, defaultSpeed_, light_sensors_);
      break;
    case kAggressive:
      bv_wheel_velocity = ab_.CalculateVelocity(
        closest_bv_entity_, defaultSpeed_, light_sensors_);
      break;
    case kLove:
      bv_wheel_velocity = lb_.CalculateVelocity(
        closest_bv_entity_, defaultSpeed_, light_sensors_);
      break;
    case kCoward:
      bv_wheel_velocity = cb_.CalculateVelocity(
        closest_bv_entity_, defaultSpeed_, light_sensors_);
      break;
    case kNone:
    default:
      numBehaviors--;
      break;
  }
…
  if (numBehaviors) {
    wheel_velocity_ = WheelVelocity(
      (light_wheel_velocity.left + food_wheel_velocity.left +
```

```
        bv_wheel_velocity.left)/numBehaviors,
      (light_wheel_velocity.right + food_wheel_velocity.right +
        bv_wheel_velocity.right)/numBehaviors,
      defaultSpeed_);
  } else {
    wheel_velocity_ = WheelVelocity(0, 0);
  }
}
```