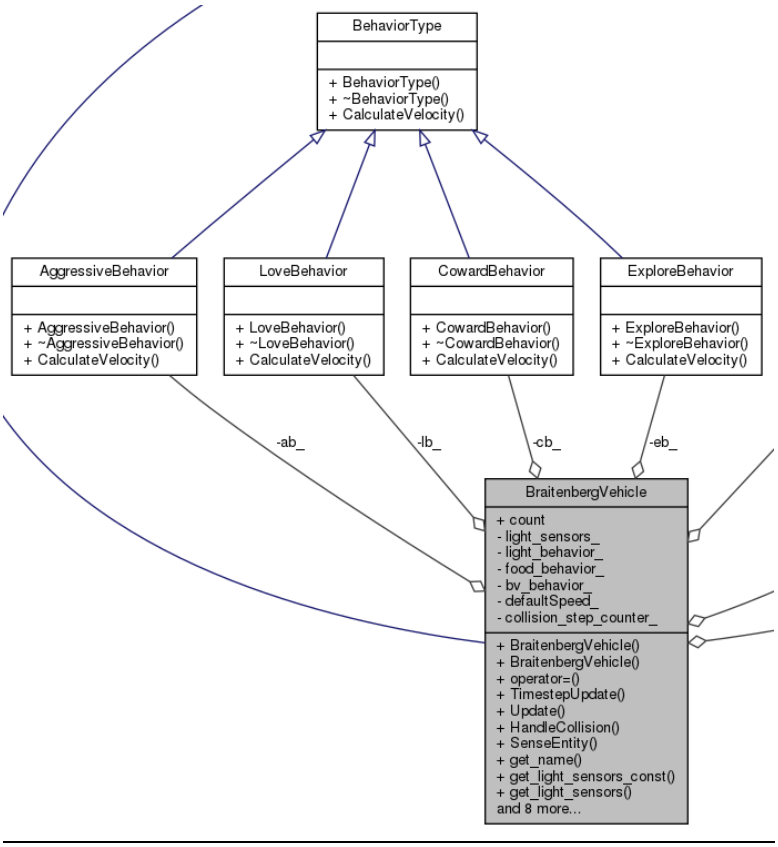
Moti Begna

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);
  }
}
```

```
/*****************************************************************************
 * Includes
 *****************************************************************************/
#include <gtest/gtest.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <streambuf>

#include "src/Aggressive.h"
#include "src/LightFactory.h"
#include "src/pose.h"

 /*****************************************************
* TEST FEATURE SetUp
*****************************************************/
class AggressiveBehaviorTest : public ::testing::Test {
public:
  virtual void SetUp() {
    light = factory.Create();
  }
protected:
  csci3081::AggressiveBehavior ab;
  csci3081::LightFactory factory;
  csci3081::Light * light;
  double speed = 5.0;
};

/*****************************************************************************
 * Test Cases
 *****************************************************************************/

TEST_F(AggressiveBehaviorTest, NoEntity) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose());
  light_sensors.push_back(csci3081::Pose());

  light = NULL;
  csci3081::WheelVelocity wv = ab.CalculateVelocity(light, speed,
     light_sensors);
```

```
  csci3081::WheelVelocity expected = csci3081::WheelVelocity(0.0001, 0.0001, speed);

  EXPECT_EQ(wv.left, expected.left) << "FAIL: Default left wheel velocity incorrectly calculted";
  EXPECT_EQ(wv.right, expected.right) << "FAIL: Default right wheel velocity incorrectly calculted";
}

TEST_F(AggressiveBehaviorTest, CloseDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 190));
  light_sensors.push_back(csci3081::Pose(200, 190));

  csci3081::WheelVelocity wv = ab.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(reading_right, reading_left, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for close distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for close distance incorrectly calculted";
}

TEST_F(AggressiveBehaviorTest, MediumDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 150));
  light_sensors.push_back(csci3081::Pose(200, 150));

  csci3081::WheelVelocity wv = ab.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(reading_right, reading_left, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for medium distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for medium distance incorrectly calculted";
```

```
}

TEST_F(AggressiveBehaviorTest, FarDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 100));
  light_sensors.push_back(csci3081::Pose(200, 100));

  csci3081::WheelVelocity wv = ab.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(reading_right, reading_left, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for far distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for far distance incorrectly calculted";
}
```

```
/*******************************************************************************
 * Includes
 ******************************************************************************/
#include <gtest/gtest.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <streambuf>

#include "src/Coward.h"
#include "src/LightFactory.h"
#include "src/pose.h"

 /*****************************************************
* TEST FEATURE SetUp
*****************************************************/
class CowardBehaviorTest : public ::testing::Test {
public:
  virtual void SetUp() {
    light = factory.Create();
  }
protected:
  csci3081::CowardBehavior cb;
  csci3081::LightFactory factory;
  csci3081::Light * light;
  double speed = 5.0;
};

/*******************************************************************************
 * Test Cases
 ******************************************************************************/

TEST_F(CowardBehaviorTest, NoEntity) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose());
  light_sensors.push_back(csci3081::Pose());

  light = NULL;
  csci3081::WheelVelocity wv = cb.CalculateVelocity(light, speed,
     light_sensors);
```

```
  csci3081::WheelVelocity expected = csci3081::WheelVelocity(0.0001, 0.0001, speed);

  EXPECT_EQ(wv.left, expected.left) << "FAIL: Default left wheel velocity incorrectly calculted";
  EXPECT_EQ(wv.right, expected.right) << "FAIL: Default right wheel velocity incorrectly calculted";
}

TEST_F(CowardBehaviorTest, CloseDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 190));
  light_sensors.push_back(csci3081::Pose(200, 190));

  csci3081::WheelVelocity wv = cb.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(reading_left, reading_right, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for close distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for close distance incorrectly calculted";
}

TEST_F(CowardBehaviorTest, MediumDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 150));
  light_sensors.push_back(csci3081::Pose(200, 150));

  csci3081::WheelVelocity wv = cb.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(reading_left, reading_right, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for medium distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for medium distance incorrectly calculted";
```

```
84  }
85
86  TEST_F(CowardBehaviorTest, FarDistance) {
87    std::vector<csci3081::Pose> light_sensors;
88    light_sensors.push_back(csci3081::Pose(200, 100));
89    light_sensors.push_back(csci3081::Pose(200, 100));
90
91    csci3081::WheelVelocity wv = cb.CalculateVelocity(light, speed,
92       light_sensors);
93
94     double reading_left =  1800.0/std::pow(
95       1.08, (light->get_pose()-light_sensors[0]).Length());
96     double reading_right = 1800.0/std::pow(
97       1.08, (light->get_pose()-light_sensors[1]).Length());
98
99     csci3081::WheelVelocity expected = csci3081::WheelVelocity(reading_left, reading_right, speed);
100
101    EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for far distance incorrectly calculted";
102    EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for far distance incorrectly calculted";
103 }
```

```
/*******************************************************************************
 * Includes
 ******************************************************************************/
#include <gtest/gtest.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <streambuf>

#include "src/Explore.h"
#include "src/LightFactory.h"
#include "src/pose.h"

 /*****************************************************
* TEST FEATURE SetUp
*****************************************************/
class ExploreBehaviorTest : public ::testing::Test {
public:
  virtual void SetUp() {
    light = factory.Create();
  }
protected:
  csci3081::ExploreBehavior eb;
  csci3081::LightFactory factory;
  csci3081::Light * light;
  double speed = 5.0;
};

/*******************************************************************************
 * Test Cases
 ******************************************************************************/

TEST_F(ExploreBehaviorTest, NoEntity) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose());
  light_sensors.push_back(csci3081::Pose());

  light = NULL;
  csci3081::WheelVelocity wv = eb.CalculateVelocity(light, speed,
     light_sensors);
```

```
  csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/0.0001, 1.0/0.0001, speed);

  EXPECT_EQ(wv.left, expected.left) << "FAIL: Default left wheel velocity incorrectly calculted";
  EXPECT_EQ(wv.right, expected.right) << "FAIL: Default right wheel velocity incorrectly calculted";
}

TEST_F(ExploreBehaviorTest, CloseDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 190));
  light_sensors.push_back(csci3081::Pose(200, 190));

  csci3081::WheelVelocity wv = eb.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/reading_right, 1.0/reading_left, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for close distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for close distance incorrectly calculted";
}

TEST_F(ExploreBehaviorTest, MediumDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 150));
  light_sensors.push_back(csci3081::Pose(200, 150));

  csci3081::WheelVelocity wv = eb.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/reading_right, 1.0/reading_left, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for medium distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for medium distance incorrectly calculted";
```

```
84  }
85
86  TEST_F(ExploreBehaviorTest, FarDistance) {
87    std::vector<csci3081::Pose> light_sensors;
88    light_sensors.push_back(csci3081::Pose(200, 100));
89    light_sensors.push_back(csci3081::Pose(200, 100));
90
91    csci3081::WheelVelocity wv = eb.CalculateVelocity(light, speed,
92       light_sensors);
93
94     double reading_left =  1800.0/std::pow(
95       1.08, (light->get_pose()-light_sensors[0]).Length());
96     double reading_right = 1800.0/std::pow(
97       1.08, (light->get_pose()-light_sensors[1]).Length());
98
99     csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/reading_right, 1.0/reading_left, speed);
100
101    EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for far distance incorrectly calculted";
102    EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for far distance incorrectly calculted";
103 }
```

```
/*******************************************************************************
 * Includes
 ******************************************************************************/
#include <gtest/gtest.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <streambuf>

#include "src/Love.h"
#include "src/LightFactory.h"
#include "src/pose.h"

 /*****************************************************
* TEST FEATURE SetUp
*****************************************************/
class LoveBehaviorTest : public ::testing::Test {
public:
  virtual void SetUp() {
    light = factory.Create();
  }
protected:
  csci3081::LoveBehavior lb;
  csci3081::LightFactory factory;
  csci3081::Light * light;
  double speed = 5.0;
};

/*******************************************************************************
 * Test Cases
 ******************************************************************************/

TEST_F(LoveBehaviorTest, NoEntity) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose());
  light_sensors.push_back(csci3081::Pose());

  light = NULL;
  csci3081::WheelVelocity wv = lb.CalculateVelocity(light, speed,
     light_sensors);
```

```
  csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/0.0001, 1.0/0.0001, speed);

  EXPECT_EQ(wv.left, expected.left) << "FAIL: Default left wheel velocity incorrectly calculted";
  EXPECT_EQ(wv.right, expected.right) << "FAIL: Default right wheel velocity incorrectly calculted";
}

TEST_F(LoveBehaviorTest, CloseDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 190));
  light_sensors.push_back(csci3081::Pose(200, 190));

  csci3081::WheelVelocity wv = lb.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/reading_left, 1.0/reading_right, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for close distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for close distance incorrectly calculted";
}

TEST_F(LoveBehaviorTest, MediumDistance) {
  std::vector<csci3081::Pose> light_sensors;
  light_sensors.push_back(csci3081::Pose(200, 150));
  light_sensors.push_back(csci3081::Pose(200, 150));

  csci3081::WheelVelocity wv = lb.CalculateVelocity(light, speed,
     light_sensors);

   double reading_left =  1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[0]).Length());
   double reading_right = 1800.0/std::pow(
     1.08, (light->get_pose()-light_sensors[1]).Length());

   csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/reading_left, 1.0/reading_right, speed);

   EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for medium distance incorrectly calculted";
   EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for medium distance incorrectly calculted";
```

```
84  }
85
86  TEST_F(LoveBehaviorTest, FarDistance) {
87    std::vector<csci3081::Pose> light_sensors;
88    light_sensors.push_back(csci3081::Pose(200, 100));
89    light_sensors.push_back(csci3081::Pose(200, 100));
90
91    csci3081::WheelVelocity wv = lb.CalculateVelocity(light, speed,
92       light_sensors);
93
94     double reading_left =  1800.0/std::pow(
95       1.08, (light->get_pose()-light_sensors[0]).Length());
96     double reading_right = 1800.0/std::pow(
97       1.08, (light->get_pose()-light_sensors[1]).Length());
98
99     csci3081::WheelVelocity expected = csci3081::WheelVelocity(1.0/reading_left, 1.0/reading_right, speed);
100
101    EXPECT_EQ(wv.left, expected.left) << "FAIL: Left wheel velocity for far distance incorrectly calculted";
102    EXPECT_EQ(wv.right, expected.right) << "FAIL: Right wheel velocity for far distance incorrectly calculted";
103 }
```