Moti Begna

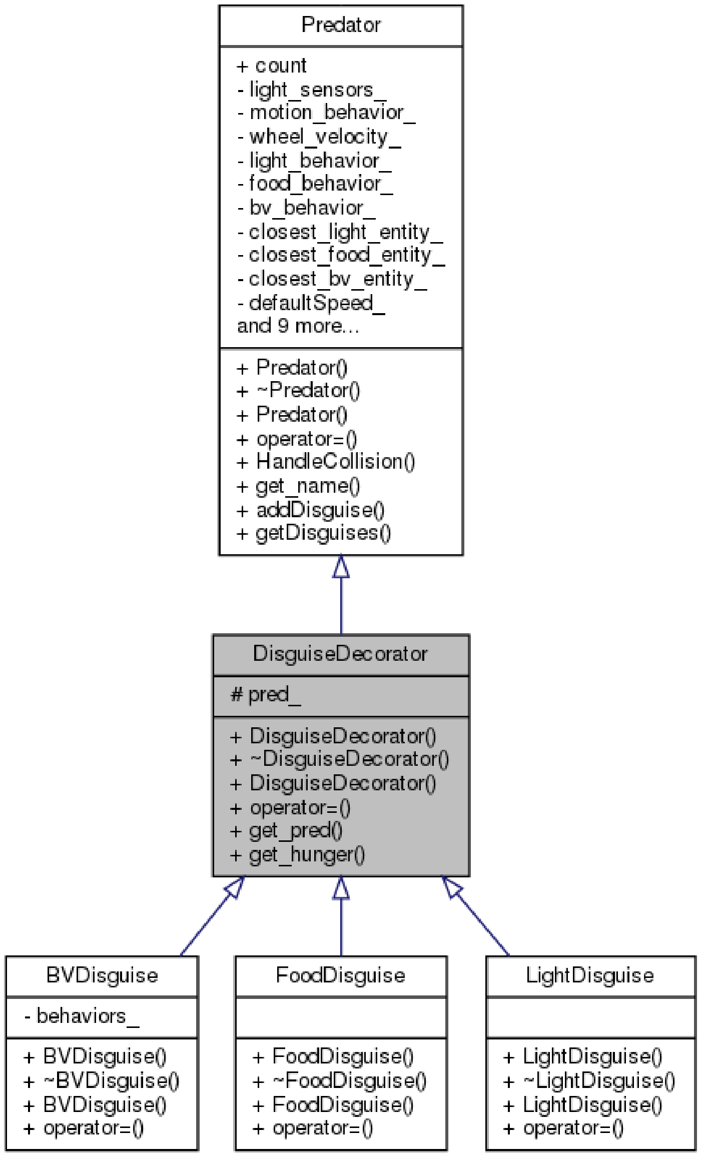
CSCI 3081

4/25/2019

Iteration 3 Preliminary 1

Doxy Generated UML for Decorator Pattern

My implementation of the Decorator pattern involves having a parent DisguiseDecorator class inheriting from a Predator, which derived disguises inherit from.



Code snippet of decorator implementation and hunger implementation

```cpp
void Arena::UpdateEntitiesTimestep() {
…
  for (auto ent : entities_) {
    ent->TimestepUpdate(1);
  }

   /* Determine if any mobile entity is colliding with wall.
   * Adjust the position accordingly so it doesn't overlap.
   */
  for (auto &ent1 : mobile_entities_) {

    // initialize disguise
    if (ent1->get_type() == kPredator){
      Predator* pred = static_cast<Predator*>(ent1);
      // increment hunger
      pred->set_hunger(pred->get_hunger() + 1);
      if (pred->get_hunger() == 150){
        int disguisetype = rand() % 3;
        pred->addDisguise(disguisetype);
        switch(disguisetype){
          case 0:
            ent1 = new FoodDisguise(pred, pred->get_hunger());
            break;
          case 1:
            ent1 = new LightDisguise(pred, pred->get_hunger());
            break;
          case 2:
            ent1 = new BVDisguise(pred, pred->get_hunger());
            break;
          default:
            break;
        }
      }
    }
    if (ent1->get_is_disguised() == true) {
      Predator* pred = static_cast<DisguiseDecorator*>(ent1)->get_pred();
      // increment hunger
      pred->set_hunger(pred->get_hunger() + 1);

      if (pred->get_hunger() == 300) {
```

```
      int disguisetype = rand() % 3;
      std::vector<int> vec = pred->getDisguises();
      // don't select a previous disguise
      while (std::count(vec.begin(), vec.end(), disguisetype)) {
        disguisetype = rand() % 3;
      }
      pred->addDisguise(disguisetype);
      // delete old disguise
      ArenaEntity* container = ent1;
      ent1 = static_cast<DisguiseDecorator*>(ent1)->get_pred();
      delete container;

      switch(disguisetype){
        case 0:
          ent1 = new FoodDisguise(pred, pred->get_hunger());
          break;
        case 1:
          ent1 = new LightDisguise(pred, pred->get_hunger());
          break;
        case 2:
          ent1 = new BVDisguise(pred, pred->get_hunger());
          break;
        default:
          break;
      }
    }

    if (pred->get_hunger() == 450) {
      int disguisetype = rand() % 3;
      std::vector<int> vec = pred->getDisguises();
      // don't select a previous disguise
      while (std::count(vec.begin(), vec.end(), disguisetype)) {
        disguisetype = rand() % 3;
      }
      pred->addDisguise(disguisetype);
      // delete old disguise
      ArenaEntity* container = ent1;
      ent1 = static_cast<DisguiseDecorator*>(ent1)->get_pred();
      delete container;

      switch(disguisetype){
        case 0:
          ent1 = new FoodDisguise(pred, pred->get_hunger());
```

```
            break;
          case 1:
            ent1 = new LightDisguise(pred, pred->get_hunger());
            break;
          case 2:
            ent1 = new BVDisguise(pred, pred->get_hunger());
            break;
          default:
            break;
        }
      }

      // kill predator aftet 600 iterations
      if (pred->get_hunger() == 600) {
        pred->set_type(kPredator);
        pred->set_is_alive(false);
      }
    }
```
…