Refactor Document

1. Replace Magic Numbers with Symbolic Constants:

   The main changes for this refactoring process are reflected in the graphics_arena_viewer.cc file. The previous implementation relied heavily on magic numbers for the slider values.

   I added new defines in the params.h file to reflect maximum count limitations for entities such as the robots, lights and food objects. Utilizing these new parameters I ensured that the default cases and the updates for the sliders would always be up-to-date with the changes in the params file. Thus allowing more customization, if params is ever changed for updated implementations.

   One of the many examples in graphics_arena_viwer:
   controller_->ChangeNumRobot((static_cast<int>(value*MAX_ROBOT)), kExplore);

   Where the MAX_ROBOT is a defined parameter in the params.h file.

2. Extract Method:

   The main changes are reflected in the robot files; robot.h & robot.cc. A new method was created that checked the hunger status of the robot, and to decrement the count of the members to ensure the robot's hunger, is working correctly.

   The method in question is called UpdateHunger, it also checks the food_flag_ which is a bool value that the arena assigns to the robot, depending on whether there are food objects in the arena.

   The TimestepUpdate method for the robot was congested, and utilizing this new method, the code became more streamlined. The following code was extracted from the robot's TimestepUpdate and added to a new method that was called during the TimestepUpdate.

```
void Robot::UpdateHunger() {
  if (food_flag_) {
    // ensure robot's death status
    if (!dead_) {
      death_timer_--;  // decrement the timer
      if (death_timer_ <= 0)
        dead_ = true;
    }
```

```
    // change the flag if the roobt is starving
    if (!is_starving_) {
      starving_--;  // decrement the timer
      if (starving_ <= 0)
        is_starving_ = true;
    }

    // change the flag if the robot is hungry
    if (!is_hungry_) {
      hungry_--;  // decrement the timer
      if (hungry_ <= 0)
        is_hungry_ = true;
    }
  }  // end outer most it
}
```

3. Change Variable and/or Method Names:

   The main changes for this refactoring is reflected in the robot.cc and robot.h files. The private members of the robot class were non-descriptive. As such the names for members including, the light and food sensors were changed to:

   - left_light_sensor_
   - right_light_sensor_
   - left_food_sensor_
   - right_food_sensor_

   These updated member names are all descriptive and self explanatory. The descriptions help the programmer understand the members easily.

4. Move Method:

   The main changes for this refactor are reflected in the motion handler robot and the robot files. The method for dealing with the Robot Behavior * from the robot was moved to the motion handler robot class, as that class is in charge of handling motion for the robot.

   The private member of the robot was also moved to the motion handler robot class, and is initialized when the robot is instantiated with a type reflected by a enum member. The robot class calls UpdateVelocity on the motion handler robot with multiple parameters including readings from all the sensors and the hunger flags, this ensures that the correct velocity is calculated, depending on the behavior of the robot assigned.

This was necessary as the motion handler robot was fairly empty, and only used for calling its parent class, and the robot class was very congested.

The following methods were extracted from robot into motion handler robot.

```
void MotionHandlerRobot::CreateBehavior(RobotBehaviorEnum behv) {
  switch (behv) {
    case kAggressive: behv_ = new AggressiveBehavior;
      break;
    case kExplore: behv_ = new ExploreBehavior;
      break;
    case kLove: behv_ = new LoveBehavior;
      break;
    case kFear: behv_ = new FearBehavior;
      break;
    default: behv_ = new FearBehavior;
  }
}
```

In addition to the code for calculing velocity, which was moved from the Robot's TimestepUpdate into the motion handler robot's UpdateVelocity method.

```
 WheelVelocity vel;
  if (starving) {
    vel.set_velocity(0.4 * food_right, 0.4 * food_left);
  } else {
    vel = behv_->Movement(light_left, light_right,
        food_left, food_right, hungry);
  }
```