Using object-oriented programming to develop simulation models of animal aggregation.

Anissa Kennedy, Shubha Shrestha, and Steven V. Viscido
Department of Biological Sciences, Winston-Salem State University, Winston-Salem, NC, USA

INTRODUCTION

• Animal social behavior is often studied using computer simulation models.
• Ordinarily, computer models are implemented using procedural programming (e.g., Warburton and Lazarus 1991, Huth and Wissel 1992, Romey 1996).
• Characteristics of procedural programming:
  • Based around variables and functions.
  • Emphasizes solving specific problems.
  • Modifications are increasingly difficult as the program grows.
• Object-oriented programming is used much less commonly to study animal aggregations.
• Characteristics of object-oriented programming (Miles and Hamilton 2006, Wiesfeld 2008):
  • Based around discrete objects.
  • Each object has its own state and procedures.
  • Modularity allows easy modification regardless of program size.
• Due to these benefits, we developed a new simulation model of animal aggregation using object-oriented programming.

Goal: The goal of this project is to use object-oriented programming to develop a useful tool for behaviorists interested in modeling the factors influencing animal behavioral patterns.

MODEL DESIGN AND DEVELOPMENT

• We used the following steps in designing our object-oriented model:
  • Step 1: We identified all the object classes needed for the model.
    • A class contains all the computer code for a particular kind of object.
    • Example: The "Flocker" class (Figure 1).
    • All animals that form groups (i.e., "flocks" or "herds") belong to this class.
  • Step 2: We defined the properties and methods for each class.
    • Properties are data stored on the object (e.g., position and velocity).
    • Methods are algorithms implemented by the object (e.g., the Movement Rule).
    • An object stores its states as properties and exhibits its behaviors through the methods.
    • Example: The Flocker class (Figure 1).
  • Step 3: We specified the relationships among related object classes (Figure 2).
    • Methods can be inherited by "child" objects.
    • Allows classes to inherit commonly used behaviors.
    • Example: RomeyFish, a subclass of Flocker, inherits the Movement Rule method (Figure 1, 2).
  • Step 4: We specified the relationship between non-related object classes.
    • Containers – objects that store other objects.
    • Containers keep track of data inside them.
    • Example: Animal objects contain "radio transmitter" objects.
  • Step 5: We used object-oriented data storage.
    • Rather than store data in a flatfile, we stored data on the objects themselves.
    • "Radio Transmitter" objects:
      • Stored inside each animal object.
      • Record all data for a simulation.
    • Statistics are stored on a "statistics" object.
    • All data were stored as Matlab objects.
    • Allowed easy access to the data – no data conversion is required.
  • Step 6: We created a global simulation script to execute the model.
    • The script simply tells each object to execute its movement rules.
    • Objects are responsible for their own movement (position, velocity, collision-avoidance, etc.).
• We used our object-oriented computer model to generate the results (Figure 5).

SIMULATION METHODS

• To demonstrate the usefulness of the object-oriented approach, we performed simulations under two conditions (Figure 3).
  • Condition 1: 60 identical fish objects all following the movement rules from Romey (1996).
• Models were otherwise run using identical parameters
  • 10 replicates of each condition
  • 500 time steps of 1 second each
  • A starting area 100 body lengths (BL) in diameter
  • Constant velocity (1 BL/s)
  • Model-specific parameters are listed in Figure 4

RESULTS

• Object-oriented design allowed for quick, efficient model implementation.
• There was a strong difference in population dispersion and group movement between homogeneous and heterogeneous populations (Figure 6).
• The heterogeneous population produced groups that were both more spread out (Figure 7, expance), and more polarized (Figure 7, polarity) than the homogeneous population model.
• The model’s outcome depended greatly on both the algorithm used and the makeup of the population (homogeneous vs. heterogeneous).

CONCLUSIONS

• Object-oriented design allowed mixed populations of organisms using different models for movement.
• Heterogeneous and heterogeneous populations behaved in different ways.
• Further research into groups of non-identical organisms is needed to understand the role of individuality in group behavior.
• Object-oriented design is an efficient way to tackle the problem of individuality.

REFERENCES


ACKNOWLEDGMENTS

We thank the many individuals who assisted in the completion of this project. Winston Hill and Denise Lea assisted with various aspects of the project. Dr. Nick Hristov provided many useful suggestions during model development. Partial funding was provided to Anissa Kennedy by the Summer Undergraduate Research Experience (SURE) program at WSSU (funded by NSF grant #0927905 to the WSSU Department of Chemistry). Primary funding was provided by NSF grant IOS-1149802 to Dr. Steven Viscido.