David Eitel
Eric Wheeler

Team 6: Flock of CGlulZ

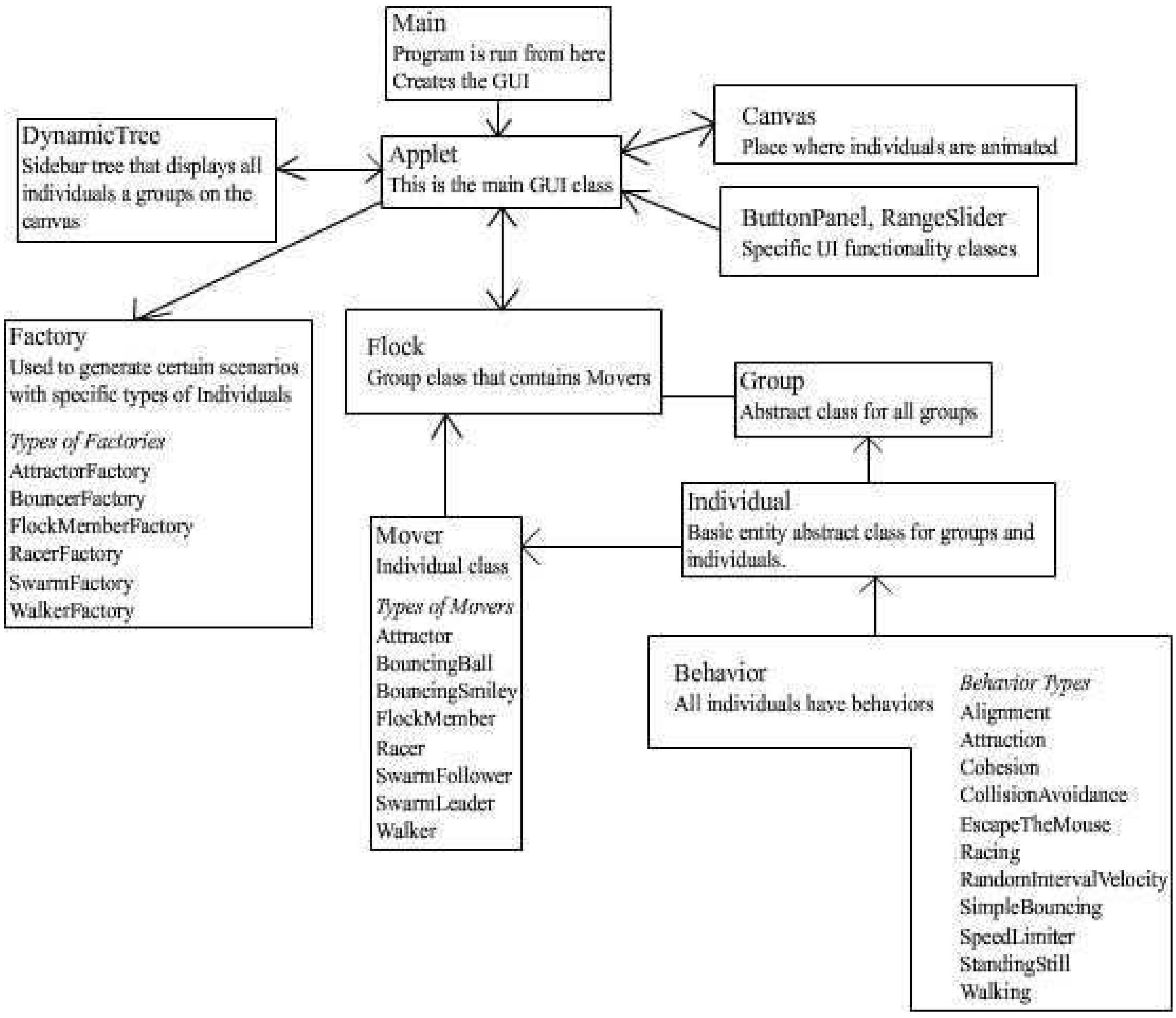
Flocks

http://www.duke.edu/~dae4/cps108/flocks/flockindex.html

Introduction

Our goal is to produce a GUI interface for driving a "game engine" of sorts. The end result should be as easily extensible as possible for different formation types, different behaviors, and dynamic updating of graphical information. We hope to get our flocks recognizing obstacles and responding to them, allowing the mouse to be recognized by the graphics to promote human interaction with the program, and allowing individual flock members some autonomy from the collective flock.

Overview



The program is initially run from the Main class. Main creates an instance of the GUI, which is an Applet. Applet is composed of many elements from our guiAndAbstracts package. Most notably, Applet includes two Button Panels, a Dynamic tree for showing the organization of the groups and individuals on the Canvas, and RangerSliders that are used to specify group/individual properties. In addition, checkboxes and radiobuttons in the JMenuBar are used to specify behaviors and group starting formations respectively.

Applet creates a Canvas on which the animation is drawn. Canvas contains the initial Flock group that is modified by the user.

Flock extends an abstract class called Group, which is the basic Group element.

The basic individual element in the program is an abstract class called Individual. All things that appear on the Canvas screen (including all groups) inherit Individual.

The general individual class used by this program is Mover. There are several types of Movers that exhibit special behaviors. Here is a list of all the classes that extend Mover:

Attractor
BouncingBall
BouncingSmiley
FlockMember
Racer
SwarmFollower
SwarmLeader
Walker

All behaviors in this program are of the type Behavior and can be found in the behaviorTypes package. Behavior types include the following:

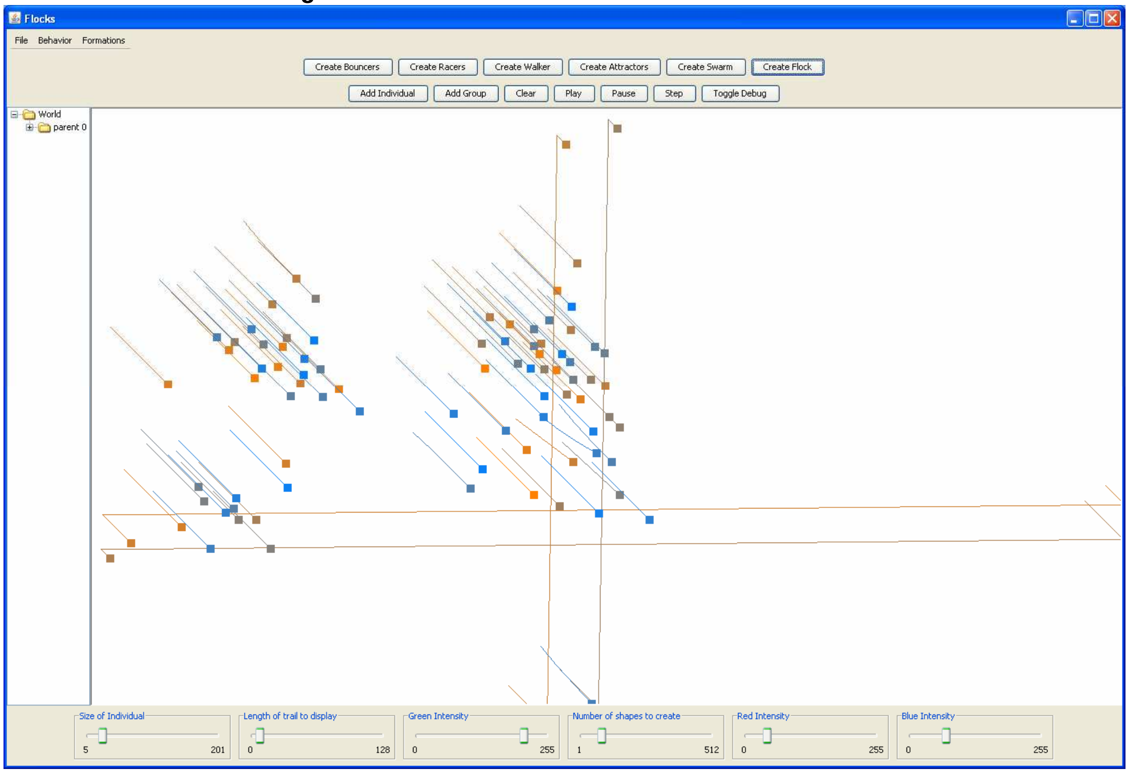
Alignment
Attraction
Cohesion
CollisionAvoidance
EscapeTheMouse
Racing
RandomIntervalVelocity
SimpleBouncing
SpeedLimiter
StandingStill
Walking

We also have Factory classes that extend an abstract Factory class. These factories are used to create specific scenarios of Movers on a Canvas. The following Factories are included with this program in the creators package:

AttractorFactory
BouncerFactory
FlockMemberFactory
RacerFactory

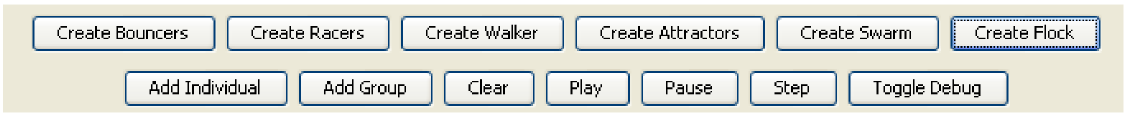
SwarmFactory
WalkerFactory

User Interface Design



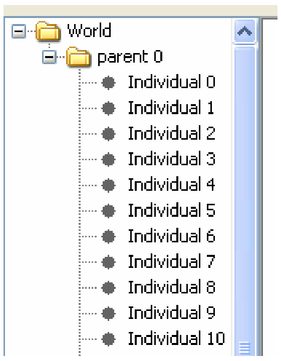
Our user interface centers around the action of creating individuals and groups of objects with specific behaviors.

We include several pre-made groups that can be added with one button press. These groups are bouncers, racers, walkers, attractors, swarms, and flocks. To create a group of one of those types, one needs to press one of the following buttons at the top of the screen:

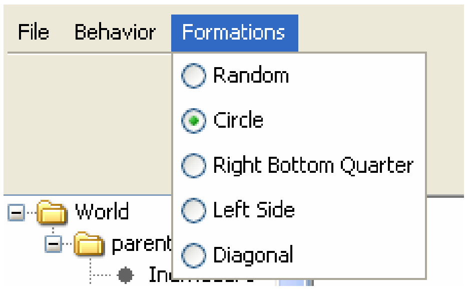


The more advanced options are underneath the “Create XXXXX” buttons.

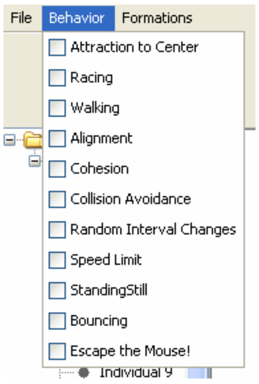
The Add Group button will add a group to the canvas. When a group is added, it will appear in the Dynamic tree menu on the left of the screen. See the below picture for an example.



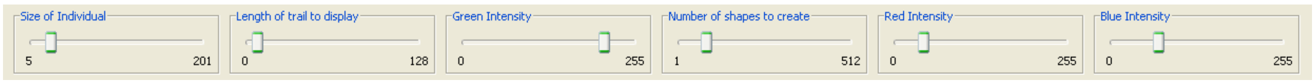
The properties of the group have to be specified by a user.
The first property that *must* be set is the starting formation of the group. This must be selected from the Formations menu. The available choices are Random (individuals start at random locations), circle, right bottom quarter, left side, and diagonal.



The next property that should be specified for a group is the behavior of individuals. Multiple behaviors can be chosen.

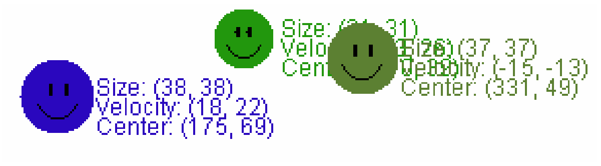


The other properties are specified by sliders at the bottom of the user interface. One slider specifies the size of individuals. Another slider determines the length of a trail to display. The other three sliders determine the RGB color of the individuals.



Add individual uses the same sliders and properties (except formations). It is supposed to add an individual to the currently selected group in the dynamic tree.

We also include buttons to Pause, Play, and Step through the animation one frame at a time. Additionally, there is a Toggle Debug button that when toggled, will display every individual’s Size, velocity, and center. See the below picture for an example.



File->Exit exits out of the program.

Design Details

Description of all behaviors:

Behavior: This is the interface for updating velocities in various ways for "movers". The interface provides extensibility for allowing a user to make a "plug-in" for any type of movement he/she wants a mover to exhibit.

Alignment: Behavior that makes a group move in a similar direction. Since each mover contains the flock that it is in, it can access the velocity vectors (and hence the directions) that the whole flock is currently moving. Flock (group of movers) has a method to figure out the average velocity vector.

Attraction: This behavior takes the "Attractor" class' original behavior and places it in the framework of our program. Attraction causes the mover to try to seek the center point of the canvas. Movers with the attraction behavior tend to not reach the exact center of the canvas I think due to rounding/casting errors when casting the calculated movement value into an integer.

Cohesion: This behavior is part of the "flock" package causing a mover to attempt to seek the center of its group. This behavior works by accessing the flock's method to find the average position and then steering the mover toward it.
CollisionAvoid: This behavior is part of the "flock" package causing a mover to attempt to avoid members of its flock that are too close to it. This behavior could easily be changed to make it avoid anything including the walls or members not of its flock. Right now it uses a flock method to get all neighboring flock members and steers away from their centers if they are within a CONSTANT radius. It would be nice to be able to dynamically change it on a slider or text field GUI value, but we did not have time to implement that functionality so we hard coded a flock radius.

EscapeTheMouse: This behavior is used in conjunction with a mouse listener on the canvas to cause movers to try to run away from the mouse's current on screen location. Other behaviors tend to overpower this one, so using a speedlimiter in conjunction with this is recommended.

Racing: This behavior is lifted from the "RACER" class. It makes the mover move to the right with different pseudo random velocities. This does not control the painting of the leader, you only get paint updates on the winner if you are using a RACER object specifically.

RandomIntervalVelocity: Adds a random unit vector to the current velocity of any mover with this velocity at a specific timing interval. Right now it adds its velocity every 12 frames. This is a constant value in the program, but it might be nice to be able to dynamically update that to see the effect of high and low random noise.

SimpleBouncing: This behavior emulates the bouncing ball behavior. Uses naive gravity and elastic collisions with all walls as of right now. It would be better to have a "Gravity" behavior that simulates complex gravitational systems in the future. Additionally, we could add on an Inelastic Collision behavior that takes into account the masses and velocities of each collider. These two behaviors together would model this situation much better.

SpeedLimiter: This behavior, as its name implies, keeps the x and y velocities below a specified threshold. In the future, one might want to split this into limitX and limitY and add a method to easily change the maximum speed because right now it is just a constant.

StandingStill: This behavior is for future implementations of NONMOVING individuals. Specifically, it naively sets the velocity of the object to zero at each update step to make sure that it will not move.

Walking: This behavior emulates the "Walker" objects behavior of yore which executes pseudoRandom movement in any direction.

Factory Discussion

Factory: The factory idea is now only useful when thought of as a "scenario editor" of sorts. Each factory as described below acts basically as a preset set of GUI values for Add Group. It always instantiates the particular formation associated with that mover type, a particular color set, a particular size and shape, as well as initial velocities and points appropriate for the "scenario" in question. For instance, a RacerFactory makes a set of Racers lined up evenly on the left hand wall with a particular behavior set (Racing). We kept the Factory idea around mostly for the utility of setting up these presets if desired. For instance, if someone wanted to make a simulator for a Rhino stampede and created the correct set of Behaviors there could be a Create Rhino scenario that showed those behaviors.

Attractor/AttractorFactory: AttractorFactory makes a number of Attractor objects set in the GUI and places them in a circular formation about the center of the canvas and provides each of them with the Attraction behavior. This creates the scenario originally seen in screen savers with N attractors swooping through the middle before coming to rest clustered about the middle.

Walker/WalkerFactory: WalkerFactory makes a number of Walker objects set in the GUI and places them in a diagonal line from left to right. It provides them the Walker behavior and places them into pseudoRandom motion wherein they will eventually spiral out from their initial positions as demonstrated in the original screen savers code.

Racer/RacerFactory: RacerFactory creates a number of Racer objects set in the GUI and places them evenly along the left wall. Racer objects are made with the Racing behavior type and given a random velocity in the positive x direction. Additional functionality for the Racer is painting the one with the largest x position red while the others remain black.

Bouncer/BouncerFactory: BouncerFactory creates a number of BouncingSmiley objects set in the GUI and places them randomly about the screen. The bouncer holds the SimpleBouncing behavior and bounces off all the walls with elastic collisions as well as using simple gravity. Each ball has a random color associated with it as well.

FlockMember/FlockMemberFactory: FlockMemberFactory creates a number of flock members set in the GUI and places them in a new flock object. FlockMembers have many behaviors including Alignment, Cohesion, CollisionAvoidance, RandomIntervalVelocity, and SpeedLimiter. FlockMembers simulate the behavior of a flock of birds that generally stick together, move in a common direction, and avoid running into each other. This is the prototypical idea of scenarios. The factory/behavior structure will allow for fairly easy implementation of both particle systems and other animal behavior types as programmed AI scenarios.

Group--Individual--Flock--Mover hierarchy

In this program the most basic element is an Individual. An Individual has a name, id, collection of behaviors, and a debug flag. We decided that these properties are required by every other thing in the program. It is worth noting that we technically implemented all of the methods specified in the real Java interface Principal. However, I did not include the implementation statement because I believed that this would be confusing to someone reading the code.

Group is the basic group entity in this program. Group extends individual, which means that a group can be treated like an individual in certain situations. This is useful for inter-flock behavior.

Because both groups and individuals appear on canvas, we included paint and update methods in both abstract classes.

We created one class that extends Group called Flock. Flock is essentially a Group of Mover objects. Mover individuals are essentially Individuals that have velocity.

It is worth noting that it is possible to create Groups within Groups because Groups contain a Collection of Individuals. Since Group extends Individuals, Groups can be placed in that Collection.

Design Considerations

Behaviors:

We designed the behavior interface to allow for an infinitely wide set of velocity updates. Each behavior has one method that takes the current velocity and updates it with new information based on the canvas state. By mixing and matching these behaviors, we can get an incredibly diverse set of behavior from our individuals on the screen. The behavior interface also makes it a snap to add an interesting set of velocity changes to a mover. All one needs to do is create a new class that implements behavior, tell it how to change velocity based on its current velocity, the objects around it, and the canvas state, and then create a new check box in the behavior GUI list for it. Checking that box off will make whatever individual or group you want to add behave in that manner. The order you check things off does matter since the object will update in the same order you clicked the boxes. Thus, if you click on stand still then racing it will behave like a very slow racer. It will not simply stand still unless stand still is the final box you checked. Behaviors also allow us to implement the user interface extension goal. By simply adding a MouseMotionListener to the canvas, we were able to get individuals to avoid running into the mouse pointer. We can also easily implement an attracting behavior to make individuals chase the mouse.

Individual/Group:

We decided against implementing the Principal and Group interfaces that are built into Java because they would be somewhat confusing to someone reading the code who is not familiar with it. Additionally, those interfaces seem more geared towards security applications.

Remaining Issues

We ran out of time so we were unable to fully implement adding Groups to Groups. However, the functionality is there because all Groups contain Individuals, and all Groups extend Individual. Therefore, Groups can contain Groups.

One needs to explicitly click a formation before pressing Add Group. Otherwise, the group is not created with the specified properties.

The IDs for all individuals and groups are not unique, though they are supposed to be unique.

Team Responsibilities

Eitel:	Getting JTree to work.
	Reflection package implementation throughout the program.
	Designing Group--Individual--Flock--Mover hierarchy.
Wheeler:	Behavior types and implementing behaviors.
	Making GUI menus and sliders.
	Making new movers access GUI for information.