Birds: Game Design

Dustin Schmidt, Karl Hiner

November 28, 2012

Game Overview

- Game Rules
  - Birds pursue and eat food
  - Hawks pursue and eat Birds
  - Birds flee Hawks
  - Player strategically places food to guide Birds to the Nest
- Scoring
  - Some number of birds reach the nest
  - Birds eat some number of food items
- AI Mechanism
  - Decisive Pathfinding
  - Allow Birds to find Food
  - Allow Hawks to find Birds
  - Allow Birds to flee Hawks

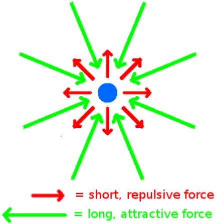
AI Overview

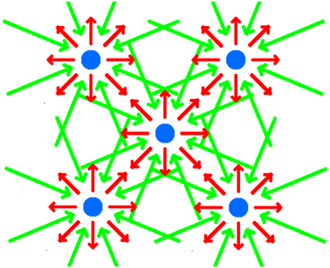
- Extension of Collaborative Diffusion
- Model reality by diffusing metrics about pursued agents throughout environment
- Agents can affect different metric layers
- Agents make decisions based on metrics in their immediate neighborhood
  - Agents compute a weighted summation of various metric layers
  - Agents of different types can weight metrics differently (including negative weights)
  - Agents move to the maximum valued cell after weighted summation

Why Use Diffusion?

- Path Finding for arbitrary number of entities / goals
- Complexity is constant as entities and goals increase
  - Diffusion is $O(nc)$, where n is the number of iterations and c is the number of cells in the map
  - Once the scent is diffused, each agent make a movement choice in $O(1)$ time
- By using multiple diffusion layers and weights, complex behaviors emerge
  - Division of Labor
  - Simultaneous Goals
  - Dynamic goal-changing based on circumstance
  - Biological Plausibility

Example: Flocking behavior





Basic algorithm

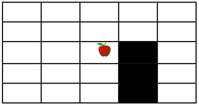
For each diffusion layer: //Food, Explore, Bird, Etc.
  $diffusionMatrix[pos.ofeveryentityinlayer] := maxDiffusion$
  $//maxDiffusion == 1$, usually. This is called the 'seed'
  All other cells are set to 0
    For NumIterations: //how many times to diffuse the entire layer
      For each x, y in $diffusionMatrix$:
        If $environment[x][y]$ is not an obstacle:
          Set $diffusionMatrix[x][y]$ = Sum of Neighboring Cells * $diffusionRate$

Diffusion Algorithm

- Sum of Neighbors Operator:
  $S(X) : S_{i,j} = X_{i+1,j} + X_{i,j+1} + X_{i-1,j} + X_{i,j-1}$
- Input for game map of size $m \times n$:
  - Metric Seed: $M_{m\times n} : M_{i,j} = \begin{cases} 0 & \text{if no metric seed exists at cell } (i,j) \\ 1 & \text{otherwise} \end{cases}$
    The originating metric values to be diffused
  - Obstacle Mask: $O_{m\times n} : O_{i,j} \in \{0,1\} \forall i < m, j < n$
    A mask to hide obstacles. Has value of 0 where obstacles exist, 1 everywhere else
  - Diffusion Matrix: $D \in \mathbb{R}_{m\times n}$
    Existing diffusion array to be further diffused. Initially all zeros
  - Diffusion Rate: $d : d \in \mathbb{R}, d < 1$
    Diffusion rate scalar, controls how much one cell bleeds into another
  - Iteration count: $i : i \in \mathbb{Z}, i > 0$
    Numer of diffusion iterations to apply. Controls rate at which metric diffuses through environment
- Return Value: Diffusion Matrix: $\hat{D} \in \mathbb{R}_{m\times n}$: Metric seed values diffused
- Precomputed Values:
  - Metric Mask: $\bar{M}_{m\times n} : \bar{M}_{i,j} = \begin{cases} 1 & \text{if } M_{i,j} = 0 \\ 0 & \text{otherwise} \end{cases}$
    A mask to hide metric seed cells.
    Need only be computed when M changes a 0 valued cell to non-zero value.
    Has value of 0 where metric seeds exist, 1 everywhere else
  - Neighbor coefficient matrix: $\hat{N} \in \mathbb{R}_{m\times n}$
    Coefficients to compute the average of neighboring cells, has value of 0 for obstacles
- Diffusion Algorithm ($\odot$ deonotes element-wise matrix multiplication):
  $\hat{D} \leftarrow D$
  FOR $j = 1 : i$
  $\quad \hat{D} \leftarrow \hat{D} + d(S(\hat{D}) \odot \bar{M} \odot \hat{N}) + M$
  RETURN $\hat{D}$

Diffusion Example

Environment:



Simple environment depicting a food item placed on a non-wrapping game map with obstacles

Metric Seed Matrix: $M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

A numeric matrix with 0 at empty cells and 1 at cells containing food

Metric Mask: $\bar{M} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

A mask to prevent modifying values in food containing cells. Helps ensure that values of diffusion array are bounded

Diffusion Example

Obstacle Mask: $O = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$ A mask to prevent diffusion through obstacle cells

Neighbor Count: $N = S(O) = \begin{bmatrix} 2 & 3 & 3 & 3 & 2 \\ 3 & 4 & 4 & 3 & 3 \\ 3 & 4 & 3 & 3 & 2 \\ 3 & 4 & 3 & 2 & 2 \\ 2 & 3 & 2 & 2 & 1 \end{bmatrix}$ A mask to prevent diffusion through obstacle cells

O/N Coefficient: $\hat{N}_{i,j} = \frac{O_{i,j}}{N_{i,j}} = \begin{bmatrix} 0.5 & 0.33 & 0.33 & 0.33 & 0.5 \\ 0.33 & 0.25 & 0.25 & 0.33 & 0.33 \\ 0.33 & 0.25 & 0.33 & 0 & 0.5 \\ 0.33 & 0.25 & 0.33 & 0 & 0.5 \\ 0.5 & 0.33 & 0.5 & 0 & 1 \end{bmatrix}$ A mask to prevent diffusion through obstacle cells

Agent Utilization of Metrics

Game Representation

Extensions and Enhancements