Birds: Game Design

Dustin Schmidt, Karl Hiner

November 30, 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

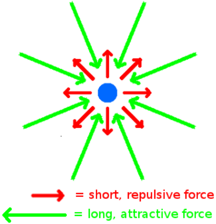
AI Overview

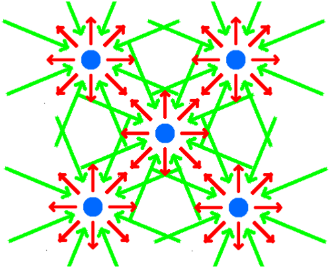
- Extension of Collaborative Diffusion
- Model reality by diffusing metrics about pursued agents throughout environment
- Agents can affect different metric layers
- Agents make decisions based on metrics in their immediate neighborhood
  - Agents compute a weighted summation of various metric layers
  - 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
- By using multiple diffusion layers and weights, complex behaviors emerge
  - Division of Labor
  - Simultaneous Goals
  - Dynamic goal-changing based on circumstance

Example: Flocking behavior





Basic algorithm

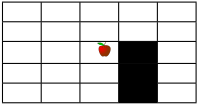
For each diffusion layer: //Food, Explore, Bird, Etc.
  $diffusionMatrix[pos.ofeveryentityinlayer] := maxDiffusion$
  $//maxDiffusion == 1$, usually. This is called the 'seed'
  All other cells are set to 0
    For NumIterations: //how many times to diffuse the entire layer
      For each x, y in $diffusionMatrix$: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

Diffusion Algorithm

- Sum of Neighbors Operator:
  $S(X) : S_{i,j} = X_{i+1,j} + X_{i,j+1} + X_{i-1,j} + X_{i,j-1} + X_{i-1,j-1} + X_{i+1,j+1} + Xi + 1, j - 1 + X_{i-1,j+1}$
- 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: $O_{m \times n} : O_{i,j} = \begin{cases} 0 & \text{for obstacle cells } (i,j) \\ 1 & \text{otherwise} \end{cases}$
    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: $c : c \in \mathbb{Z}, c > 0$
    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
  - NeighborCoefficient: $n = 8$ count of neighboring cells used in diffusion
- Diffusion Algorithm ($\odot$ deonotes element-wise matrix multiplication):
  FOR $k = 1 : c$
  $\quad \hat{D} \leftarrow \frac{d}{n}(S(D) \odot \bar{M}) + M$
  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

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

Agent Utilization of Metrics

- Agents are given an array of metric matrices representing the metrics of interest in their neighborhood of the game map
- Neighborhood is an array of 3×3 matrices
- Agents apply a scalar weight to each layer representing their strategy
- Agents perform element-wise sum the weighted matrices
- Agents select the maximum valued cell as their destination

Game Representation - Behavior

```
'Metrics':{
 'FoodMetric':{'rate':0.2,'iters':1},
   'BirdMetric':{'rate':0.9,'iters':15},
   'HawkMetric':{'rate':0.7,'iters':15},
   'NestMetric':{'rate':0,  'iters':0}
  },
'Entities':{
'Food':{ 'Eats':[],
 'Weights':{},
 'MapChar':'F',
 'Image':'apple.png',
 'Affects':{'FoodMetric':1},
 'Moves':False,
 'StartSkill':0
},
'Bird':{ 'Eats':['Food'],
 'Weights':{
  'FoodMetric':1,
    'BirdMetric':0,
    'HawkMetric':-1
    },
 'MapChar':'B',
 'Image':'bird.png',
 'Affects':{'BirdMetric':'skill'},
 'Moves':True,
 'StartSkill':1
},
'Hawk':{ 'Eats':['Bird', 'Food'],
 'Weights':{
  'FoodMetric':1,
    'BirdMetric':1,
    'HawkMetric':0
    },
 'MapChar':'H',
 'Image':'hawk.png',
 'Affects':{'HawkMetric':1},
 'Moves':True,
 'StartSkill':0
},
'Nest':{ 'Eats':['Bird'],
 'Weights':{},
 'MapChar':'N',
 'Image':'nest.png',
 'Affects':{'NestMetric':'skill'},
 'Moves':False,
 'StartSkill':0
  }
},
 'InsertEntity':[{'entity':'Food','label':'food','count':15},{'entity':'Bird','label':'bird','count':12}, {'entity':'Hawk','lab
 'StartPaused':True,
 'Win':{'NestMetric':5}
```

Game Representation - Map

```
000000000000000
0.............0
0......B......0
0.............0
0.............0
0...H.........0
0........0....0
0........0....0
0.......F0....0
0........0....0
0........0....0
0........0....0
0........0....0
0........0....0
0........0....0
000000000000000
```

Extensions and Enhancements

- Referencing Affected Layers - Subtract own effect
- Extend to 3d
- Continuous movement over discrete map
- Genetic algorithm to evolve weights