Graphs, Search, Pathfinding
(behavior involving where to go)

Steering, Flocking, Formations
(behavior involving how to go)
Class N-2

1. What are some benefits of path networks?
2. Cons of path networks?
3. What is the flood fill algorithm?
4. What is a simple approach to using path navigation nodes?
5. What is a navigation table?
6. How does the expanded geometry model work? Does it work with map gen features?
7. What are the major wins of a Nav Mesh?
8. Would you calculate an optimal nav-mesh?
1. When might you precompute paths?
2. This is a single-source, multi-target shortest path algorithm for arbitrary directed graphs with non-negative weights. Question?
3. This is a all-pairs shortest path algorithm.
4. How can a designer allow static paths in a dynamic environment?
5. When will we typically use heuristic search?
6. What is an admissible heuristic?
7. When/Why might we use hierarchical pathing?
8. Does path smoothing work with hierarchical?
9. How might we combat fog-of-war?
(kinematic) Movement, Steering, Flocking, Formations

2016-05-31
Basics

• Movement calculation often needs to interact with the “Physics” engine
  – Avoid characters walking through each other or through obstacles

• Traditional: *kinematic movement* (not dynamic)
  – Characters move (often at fixed speed) instantaneously
  – No regard to how physical objects accelerate or brake
  – Output: direction to move in

• Newer approach: *Steering behaviors* or dynamic movement (Craig Reynolds) –
  – Characters accelerate and turn based on physics
  – Take current motion of character into account
  – Output: forces or accelerations that result in velocity change
  – flocking ⊂ steering

http://www.cse.scu.edu/~tschwarz/coen266_09/PPT/Movement%20for%20Gaming.ppt
General Algorithm

Millington Fig 3.2
Assumptions

• Computed quickly
• Impression of intelligence, not a simulation
• Character position model: point + orientation
• Full 3D usually unnecessary (ie scalar $\Theta$)
  – 2D suffices, thanks to gravity
    • $(x, y, \Theta)$ ... 3 degrees of freedom
  – $\frac{3}{2}$ D (3D position, 2D orientation) covers most
    • $(x, y, z, \Theta)$ ... 4 degrees of freedom
• Rotation is the process of changing orientation
Space

• Axes
• Orientation
• Local vs global coordinate systems

Character is at
$x = 2.2$
$z = 2$
orientation = 1.5

1.5 radians

Millington Fig 3.4
Vector Form of Orientation

• Convenient to represent orientation as unit vector (|v| = 1)

• $\vec{w}_v = [\sin \alpha_s, \cos \alpha_s]$
Statics

• Static, because no information about movement
  – Position
    • 2 or 3-dimensional vector
  – Orientation
    • 2-dimensional unit vector given by an angle, a single real value between 0 and 2 $\pi$
Kinematics

• We describe a moving character by
  – Position: 2 or 3-D vector
  – Orientation
    • 2-dimensional unit vector given by an angle, a single real value between 0 and 2 $\pi$
  – Velocity (linear velocity): 2 or 3-D vector
  – Rotation (angular velocity)
    • 2-dimensional unit vector given by an angle, a single real value between 0 and 2 $\pi$

• Movement behaviors output
  – Velocity
  – Rotation
**Time & Variable Frame Rates**

- Velocities are given in units per second rather than per frame
- Older games often used per-frame velocity
- Explicit update time supports VFR. E.g:
  - character going 1 m/s
  - Last frame was 20ms duration
  - Next frame, character moves 20 mm
Kinematics

• Computing a new target velocity based on \( \{x,z\} + \phi \) can look unrealistic
  – Can lead to abrupt changes of velocity
  – Must smooth velocity, or use acceleration model

• \( \{x,z\} + \phi + v \rightarrow \) can increment velocity by some \( \Delta \) from \( \text{curr}_v \) up to \( \text{target}_v \)

• Must track velocity in all dimensions plus rotation
Facing

- Motion & facing need not be coupled
- Many games simplify & force character orientation to be in direction of the velocity
  - Instant (can be awkward)
  - Smoothing

Millington Fig 3.6
Changing Orientation

• Uses static data (position & Θ, no velocity)
• Outputs desired velocity
  – On/off in target direction
  – Smoothing may be done (without \( a \))
• New \( v \) determines new \( \Theta \)
  – If \( v > 0 \), return \( \text{atan2}(-\text{static}.x, \text{static}.z) \)
  – Else use current orientation
Kinematic Seek

• Input: character’s & target’s static data
• Output: velocity in direction from *char* to *targ*
  ▪ velocity = target.position – character.position
• Normalize velocity to maximum velocity
• Can ignore orientation, or update (prev slide)
• Flee = character.position – target.position
• O(1) in time and memory
Kinematic Arrival

• Seek with full velocity leads to overshooting
  – Arrival modification
    • Determine arrival target radius
    • Lower velocity within target for arrival

```cpp
steering.velocity = target.position - character.position;
if(steering.velocity.length() < radius) {
    steering.velocity /= timeToTarget;
    if(steering.velocity.length() > MAXIMUMSPEED)
        steering.velocity /= steering.velocity.length();
}
else
    steering.velocity /= steering.velocity.length();
```

http://www.cse.scu.edu/~tschwarz/coen266_09/PPT/Movement%20for%20Gaming.ppt
Kinematic Wander

• Move in current direction at max speed
• Vary orientation by some random amount each frame

Millington Fig 3.7
See also

• M website: www.ai4g.com
  – Algorithms for K {wander, arrive, seek, flee}
  – https://github.com/idmillington/aicore
• B Ch 3 (B Ch 1)
• Animations (for simple)
  – http://www.red3d.com/cwr/steer/
• http://en.wikipedia.org/wiki/Radian
Steering Behaviors (Dynamic)

• Steering extends kinematic movement by adding acceleration and rotation
  – Remember:
    • \( p(t) \) – position at time \( t \)
    • \( v(t) = p'(t) \) – velocity at time \( t \)
    • \( a(t) = v'(t) \) – acceleration at time \( t \)
  – Hence:
    • \( \Delta p \approx v \)
    • \( \Delta v \approx a \)

• Steering behaviors output accelerations
  – Linear acceleration: 2 or 3-D vector
  – Angular acceleration: single float value
Kinematic Updates

• def update(steering, time)
  – Assume frame rate is high enough
  – Steering is given as
    • Steering.Linear – a 2D vector
      – Represents changes in velocity (linear acceleration)
    • Steering.Angular – a real value
      – Represents changes in orientation (angular acceleration)
  – Update at each frame (Newton-Euler-1)
    • Position += Velocity * Time
    • Orientation += Rotation * Time
    • Velocity += Steering.Linear * Time
    • Rotation += Steering.Angular * Time
Dynamic Movement

• Dynamic movement update
  – Accelerate in direction of target until maximum velocity is reached
  – If target is close, lower velocity (Braking)
    • Negative acceleration is also limited
  – If target is very close, stop moving

• Dynamic movement update with Physics engine
  – Acceleration is achieved by a force
  – Vehicles etc. suffer drag, a force opposite to velocity that increases with the size of velocity
    • Limits velocity naturally
Steering Input Basics

• Input: agent kinematic and target info
  – Target collision info
  – Target trajectory
  – Target location
  – Average flock information

• Steering behavior doesn’t attempt to do much
  – Each alg. Does a single thing. Fundamental behaviors
  – Combine simple behaviors to make complex
  – No: avoid obstacles while chasing character and making detours to nearby power-ups
Steering Behaviors

• Variable Matching
  – Seek (flee)
  – Arrive (leave)
  – Align
  – Velocity Matching

• Best way to get a feel: run steering behavior program from source www.ai4g.com
  – https://github.com/idmillington/aicore
Variable Matching

• Simplest family: match one or more elements of source to target
  – Match **position** (seek): accelerate toward target, decelerate once near
  – Match **orientation** (align): rotate to align
  – Match **velocity**: follow on a parallel path, copy movements, stay fixed distance away

• Match position and orientation? Ok
• Match position and velocity? Conflict
• Moral: have individual matching algorithms, and conflict-resolving combination algorithm
Basic Steering Behaviors

• Used as elements of more complex behaviors
  – Pursue = Seek based on target motion (instead of position)
  – Collision avoidance = flee based on obstacle proximity
  – Wander = Seek some fictitious moving object
Obstacle and Wall Avoidance

• Cast one or more (distance-bounded) rays out in direction of motion
• Use collisions to create sub-target for avoidance
• Perform basic seek on sub-target

Millington Fig 3.24
One is not enough

Millington Fig 3.25 & 3.26
Dynamic Seek

• Match position of character with the target
• Like kinematic seek, find direction to target and go there as fast as possible
  – Kinematic outputs: velocity, rotation
  – Dynamic output: linear and angular acceleration
• Kinematic seek:
  – velocity = target.position – character.position
  – velocity = (velocity.normalize())*maxSpeed
• Dynamic seek:
  – acceleration = target.position – character.position
  – acceleration = (acceleration.normalize())*maxAcceleration
Composite Behaviors

• Pursue = Seek based on target motion (instead of position)
• Evade?
• Face?
• Looking wher going?
• Wander?

Millington Fig 3.11 & 3.12
Composite Behaviors

- Pursue / Evade
- Face / Look where going
- Wander
- Collision Avoidance
- Obstacle Avoidance
- Separation

Millington Fig 3.29
Combining Steering Behavior

• (Weighted) Blending
  – Execute all steering behaviors
  – Combine results by calculating a compromise based on weights
    • Example: Flocking based on separation and cohesion

• Arbitration
  – Selects one proposed steering

• Not mutually exclusive

• Emergent Behavior
Weighted Blending

• Simplest way to combine steering behaviors
• Weighted linear sum of accelerations from all involved steering behaviors
• Post-processing velocity threshold
• E.g. rioting crowd may have $1 \times \text{sep} + 1 \times \text{cohes}$
• Finding “right” weight can be challenging
  – Characters can get stuck (equilibrium)
  – Constrained environments (conflicts)
  – Jitter
Millington Fig 3.35 & 3.36
Flocking and Swarming

• Craig Reynolds’s “boids” (Flocking != Swarming)
  • Simulated (apparent behavior of) birds, 1986
  • Blends three steering mechanisms (ordered)
    – Separation
      » Move away from other birds that are too close
    – Cohesion
      » Move to center of mass of flock
    – Alignment
      » Match orientation and velocity of flock
• Equal Weights for simple flocking behavior
Won’t you be my neighbor

Cohesion

Result

Match velocity/align

Separation

Center of gravity

Average velocity

Millington Fig 3.31

Millington Fig 3.32

Buckland Fig 3.18
Recall findNearestWaypoint()

- Most engines provide a rapid “nearest” function for objects
- Spatial partitioning w/ special data structures:
  - Quad-trees (2d), oct-trees (3d), k-d trees
  - Binary space partitioning (BSP tree)
  - Multi-resolution maps (hierarchical grids)
- The gain over all-pairs techniques depends on number of agents/objects
Separation

• Steer to avoid crowding local flockmates
  – Neighborhood is a sphere of certain radius, or possibly a cone of perception

http://www.red3d.com/cwr/boids/
Cohesion

- Steer to average **position** of local flockmates

* Center of mass is the average position (X,Y,Z) of boids in neighborhood.

http://www.red3d.com/cwr/boids/
Alignment

• Steer towards average **heading**

* Average heading and velocity of other boids in neighborhood

http://www.red3d.com/cwr/boids/
Buckland Fig 3.16

Separation

Alignment

Cohesion
Flocking Demos

See Also

• M Ch 3, B Ch 3 (& Ch 1)
• Source from Millington
  – https://github.com/idmillington/aicore
• Java-based animations (combined behaviors)
  – http://www.red3d.com/cwr/steer/
• http://www.cse.scu.edu/~tschwarz/coen266_09/PPT/Movement%20for%20Gaming.ppt
Formations

• Coordinated Movement: M Ch 3.7
• Path plan for leader (naive)
  – All others move toward leader
• Replace team with a virtual bot
  – All members controlled by a joint animation
• Path plan for leader (alt)
  – All team members path plan to an offset
  – Flow around obstacles and through choke points
Fixed Formations

- V or Finger
- Line
- Defensive circle
- Two abreast in cover
Entities of different sizes

- Large entities can be surrounded by smaller entities and collision boxes can parallelize large entities
- Path plan without regard
- Send messages for smaller entities to move out of the way
Next Class

- Finite state machines
- Read: Buckland, CH 2 (M Ch 5.1-4)