Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

Graphs, Search, Pathfinding
(behavior involving \textbf{where} to go)

Static, Kinematic, & Dynamic Movement;
Steering, Flocking, Formations
(behavior involving \textbf{how} to go)
Announcements

• HW 3 due Sunday night, September 22
• “Judder”/”Jidder”: shaky movement when steering behaviors conflict
• Swarming:
  – Ant colony optimization lib for py:
    • https://pypi.python.org/pypi/ACO-Pants
• BSP: http://game-ai.gatech.edu/sites/default/files/documents/assignments/bsp.html

Buckland Fig 3.19
PREVIOUSLY ON...
N-2: Movement & Steering

1. What do movement algorithms output in static environ?
2. What do movement algorithms input in kinematic environ?
3. What do movement algorithms output in kinematic environ?
4. What is the deal with time & variable frame rates?
5. What was the insight about updates if time << 1?
6. How are kinematic seek and pursue different?
7. What's the point of kinematic arrival?
8. Kinematic wander varies what randomly?
9. What’s the main difference between kinematic and steering/dynamic movement?
1. Steering vs flocking vs swarming?
2. Steering Family Tree
3. How might we combine behaviors?
4. Can we be sure combinations work?
5. What three steering mechanisms enable flocking?
6. Spatial partitioning w/ special data structures: Why? How?
Clarifications

- Two tier nav: local vs global
  - use steering on local
  - perform seek on sequence of path nodes for global
- acceleration & forces vs instant velocity changes
- avoiding obstacles while seeking: more discussion
Movement & Steering 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 (instantaneous change to velocity with magnitude)
• 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
struct StaticState:
  position # 2D vector
  orientation # single float

struct StaticMovementOutput:
  position # 2D/3D vector
  orientation # single float
Kinematics

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

• Movement behaviors output
  – Velocity
  – Rotation

• Movement behaviors input STATIC data
  – Position and orientation, no velocities
struct KinematicState:

  position # 2D/3D vector
  orientation # single float
  velocity # 2D/3D vector
  rotation # single float

Note: rotation is angular velocity

struct KinematicOutput:

  velocity # 2D/3D vector
  rotation # single float rps

Note: Kinematic movement algorithms only input position and orientation, output desired velocity
Direction and Distance

- Normalize for unit vector
- Magnitude for distance

\[
\begin{align*}
2D: |v| &= \sqrt{x^2 + y^2} \\
3D: |v| &= \sqrt{x^2 + y^2 + z^2}
\end{align*}
\]
Simple Movement

- Orient agent velocity in target direction
- Very aggressive and doesn’t look very natural (can change dir instantaneously)
- Fine for discrete movement (on a grid)
Steering Movement

• Turn towards target vector
• Adjust speed, perhaps slowing forward velocity if target angle is large and accelerating to max velocity as target angle becomes small
Steering Behavior: Performance Envelope

- Typically: constant acceleration and enforced max velocity for translations and rotations (speed and turn rates)
- Can get progressively more advanced, introducing acceleration curves, separate deceleration rates, speed dependent turn rates, etc.
- Can be based on forces/torques, and agent mass
Kinematic Seek & Flee

• directs an agent toward a target position
• Input: static data of character & target
• Output: velocity in direction from char to targ
  ▪ velocity = target.position – character.position
• Normalize velocity to 1 and multiply by maximum velocity
• Can ignore orientation, or update to face movement direction
• O(1) in time and memory
• Flee = -1 * velocity = character.position – target.position
Identify target

• Calculate relative position vector. A is target, B is ‘me’
  – Velocity = target.position – character.position

\[ \mathbf{r}_{BA} = \mathbf{r}_A - \mathbf{r}_B \]
\[ = (x_A - x_B)i + (y_A - y_B)j + (z_A - z_B)k \]
\[ = -\mathbf{r}_{BA} \]
Prediction (Pursue/Evade)

- No need to calculate precise intercept
  - Recalculating every frame anyways
- Dist = (target.pos – char.pos).Length()
- lookAheadT = Dist/char.maxSpeed
- futureTarget = target.pos + lookAheadT * target.velocity
- //Steer towards futureTarget
Prediction Gotchas

- Watch out for extreme predictions (very large lookahead $t$ values)
- Could be sending your agent off the map or result in odd behavior
- Consider clamping max time prediction (and even minimum)
- Consider clipping extrapolated future positions to fit on navmesh or map, etc.
Kinematic Arrival

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

```java
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();
```

Millington 3.2.1

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
  - randomBinomial() = rand() – rand(), where rand returns [0,1]
  - random number between −1 and 1, where values around zero are more likely

Millington Fig 3.7  Buckland Fig 3.4
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 behavior “zoo”
  – Combine simple behaviors to make complex
  – No: avoid obstacles while chasing character and making detours to nearby power-ups
Dynamic/Steering Output

struct KinematicState:
  position  # 2D/3D vector
  orientation  # single float
  velocity  # 2D/3D vector
  rotation  # single float

Note: rotation is angular velocity

struct SteeringOutput:
  linear_acc  # 2D/3D vector
  angular_acc  # single float
Dynamic Movement Updates

• Dynamic movement update
  – Accelerate in direction of target until maximum velocity is reached
  – (Optional) If target is close, lower velocity (Braking)
    • Negative acceleration is also limited
  – (Optional) 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
Updates to Position & Orientation

- steering.linear: a 2D vector
  - Represents changes in velocity (linear acceleration)
- steering.angular: a real value
  - Represents changes in orientation (angular acceleration)

```
def update(steering, time)
  - Update at each frame (if time << 1, use Newton-Euler-1)
  - Position += Velocity * Time + 0.5 * steering.linear * time * time
  - Orientation += Rotation * Time + 0.5 * steering.angular * time * time
  - Velocity += steering.linear * Time
  - Rotation += steering.angular * Time
```

Don’t forget to check for speeding!
If Velocity.length() > maxSpeed:
  velocity.normalize()
  velocity *= maxSpeed

Don’t forget to assign new orientation as modulo (2 \pi)
Core Steering Behaviors

• Variable Matching
  – Seek (flee): position of target
  – Align: orientation of target
  – Arrive (leave(flee)): velocity of target
  – Velocity Matching: flocking

• Best way to get a feel:
  – Look at pseudo-code in Millington & Funge
  – run steering behavior program from source [www.ai4g.com](http://www.ai4g.com), [https://github.com/idmillington/aicore](https://github.com/idmillington/aicore)
Derived & Composite Steering Behaviors

• More complex behaviors derived from core
  – Pursue (evade): Seek (flee) based on predicted target position
  – Face: Align to target orientation
  – Look where going: Face in direction of movement (using Align)
  – Collision avoidance: Flee based on obstacle proximity
  – Wander: Seek + Face some fictitious moving object
Dynamic Seek

- 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
desired_velocity = normalize (position - target) * max_speed
steering = desired_velocity - velocity

http://www.red3d.com/cwr/steer/gdc99/
(static, kinematic, dynamic) Movement

Steering Continued, Flocking, Formations

2019-09-11
M&F 3.1-3.4
B 3
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
- Alternatives:
  - Avoidance force
  - Flow field
Obstacle Test – Single Ray Falls Short

• No hard and fast rules as to which is better

- Parallel side rays
- Central ray with short fixed length whiskers
- Whiskers only
- Single only

Often good start
The Corner Trap

• multi-ray wall avoidance can get stuck on corners
• What happens below?

M&F 3.27
Flow/Force/Vector Fields

- Motion specification without use of programming (can be used by art staff directly)
- Character steers to align its motion with the local tangent of field
  - Future position of a character is estimated, and flow field is sampled at that location.
  - This flow direction (vector F) is the “desired velocity”
  - The steering direction (vector S) is the difference between the current velocity (vector V) and the desired velocity (vector F)

http://www.red3d.com/cwr/steer/gdc99/
Other approaches worth noting

- Smart maps / smart environments
  - choke points are particularly problematic: kitchen door in restaurant
  - Navigation fields provide authorial control
- Lane formation
Composite Behaviors

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

Millington Fig 3.29
Variable Matching Conflicts

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

Millington Fig 3.35 & 3.36
Combine Steering Behaviors: Problem

• What if steering behaviors are opposed?
• Zero vector!
• Or back and forth forever!
• Or orbiting!
• Need higher level control logic!

[Diagram showing exactly aligned forces balance out in dead end]
Millington Fig 3.35 & 3.36

can’t avoid an obstacle and chase

Missing a narrow doorway

Wall avoidance

Result

Pursue

Pursued enemy

Resulting acceleration

Collision ray

Route of character

Target
Combining Steering Behavior

- Sum (w/ max speed enforced)
- (Weighted) Blending
  - Execute all steering behaviors
  - Combine results by calculating a compromise based on weights
    - Example: Flocking based on separation and cohesion
- Fixed priorities
- 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
Multiple Steering Goals? Combining steering behaviors

[Diagram showing the concept of combining steering behaviors with arrows pointing to different directions and equations representing the resulting steering force.]
(static, kinematic, dynamic) Movement
Steering, **Flocking**, Formations

2019-09-16
M&F 3.1-3.4
B 3

http://www.red3d.com/cwr/steer/gdc99/
https://en.wikipedia.org/wiki/Swarm_behaviour
They’re flocking this way

https://www.youtube.com/watch?v=nM-RPO10aPY

https://www.youtube.com/watch?v=QbUPfMXQlY
Problems

• Bunching:
  – https://www.youtube.com/watch?v=ZlAmoRsu3Z0&feature=youtu.be&list=PLxGbBc3OuMgg7OuyLfVXQLR6HKcogICfG&t=1833
  – AIIDE 2015 keynote, Adam Noonchester. “AI In The Awesomepocalypse”
  – Sunset Overdrive: IGN 9.0 “Awesome”, Editors’ choice

• Too close:
  – https://youtu.be/ZIAmoRsu3Z0?list=PLxGbBc3OuMgg7OuyLfVXQLR6HKcogICfG&t=1713
Flocking and Swarming

• Craig Reynolds’s “boids” (Flocking != Swarming)
  • https://www.youtube.com/watch?v=86iQiV3-3IA
  • https://www.youtube.com/watch?v=QbUPfMXXQlY
  • 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
But 1st: won’t you be my neighbor
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
  - Brute force: $O(n^2)$  //every boid to each other
Boids – bin-lattice

• Spatial sub-division
• When boid moves, check to see if it is in a new bin (update accordingly)
• $O(n \, k)$ – $k$ is number of surrounding bins to consider

https://pdfs.semanticscholar.org/4183/faeef708a56b988742b5572fce9174caec7b.pdf
Demo

• Another big shoal (neighbors)
Separation

• Steer to avoid crowding local flockmates
  – Force to steer a bot away from neighbors
  – Neighborhood is a sphere of certain radius, or possibly a cone of perception
  – The vector to each bot under consideration is normalized, divided by the distance to the neighbor, and added to the steering force.

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

- Steer to average **position** (center of mass) of local flockmates
  - Desired position (center of mass): iterate through all neighbors and average their positions
  - Seek to desired position

* 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**
  – Attempts to keep bots aligned with neighbors
  – Desired heading: iterate through all neighbors and average their heading vectors
  – For each neighbor, subtract bot’s heading from desired heading

* Average heading and velocity of other boids in neighborhood
  http://www.red3d.com/cwr/boids/
Demo

- Flocking (turn on and off)
- Big shoal (blending other behaviors)
Flocking Demos

• http://www.red3d.com/cwr/boids/
• http://www.red3d.com/cwr/boids/applet/
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