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 **where** to go)

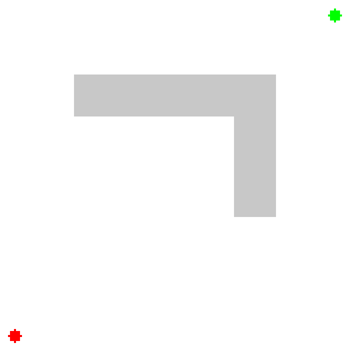
Static, Kinematic, & Dynamic Movement; Steering, Flocking, Formations

(behavior involving **how** to go)

PREVIOUSLY ON…

Graph Search: Sorting Successors

- Uninformed (all nodes are same)
  - DFS (stack – lifo), BFS (queue – fifo)
  - Iterative-deepening (Depth-limited)
- Informed (pick order of node expansion)
  - Greedy Best First
  - Dijkstra – guarantee shortest path ($E\log_2 N$)
  - Floyd-Warshall
  - A* (IDA*)…. Dijkstra + heuristic, Memory Bounded A*
  - D*
- Hierarchical can help



http://en.wikipedia.org/wiki/A*_search_algorithm

N-1: Search recap

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?

(static, kinematic, dynamic) Movement Steering, Flocking, Formations

2019-09-11

M&F 3.1-3.4

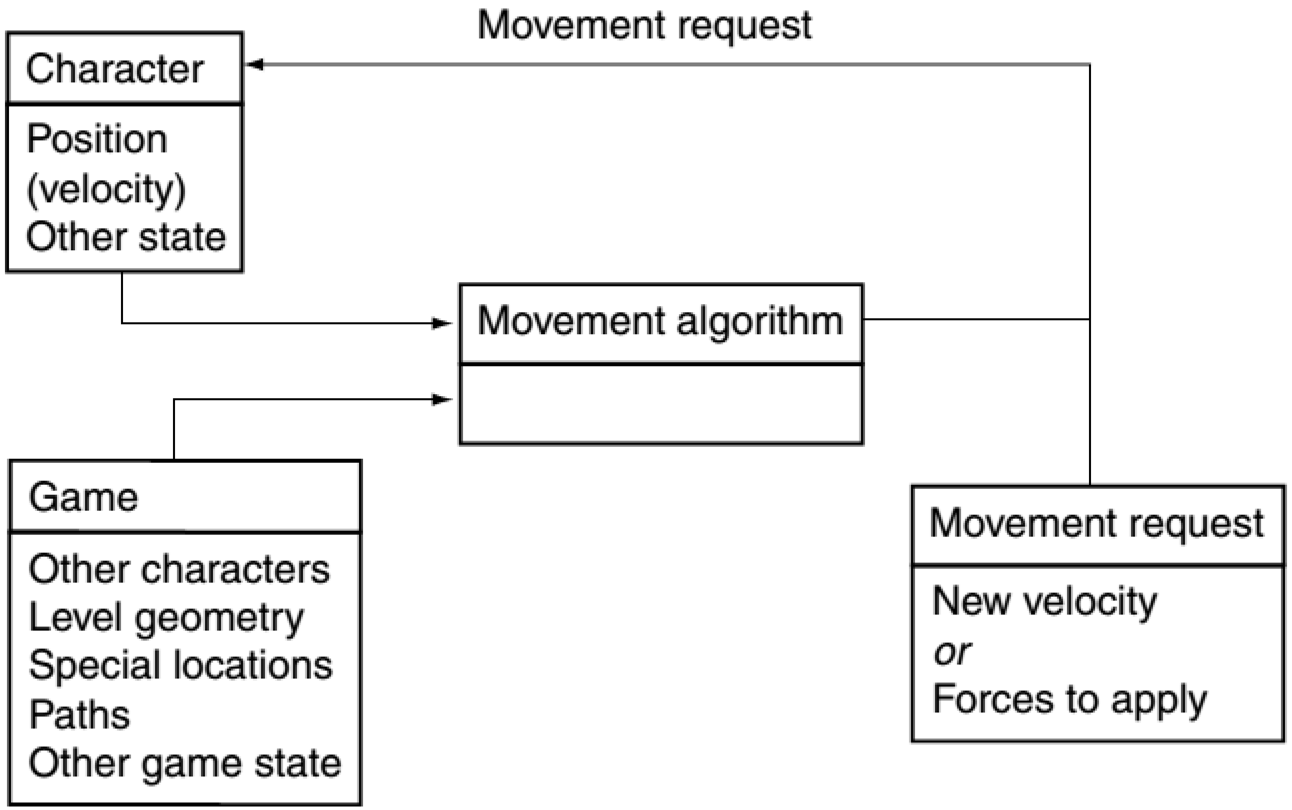
B 3

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

General Algorithm



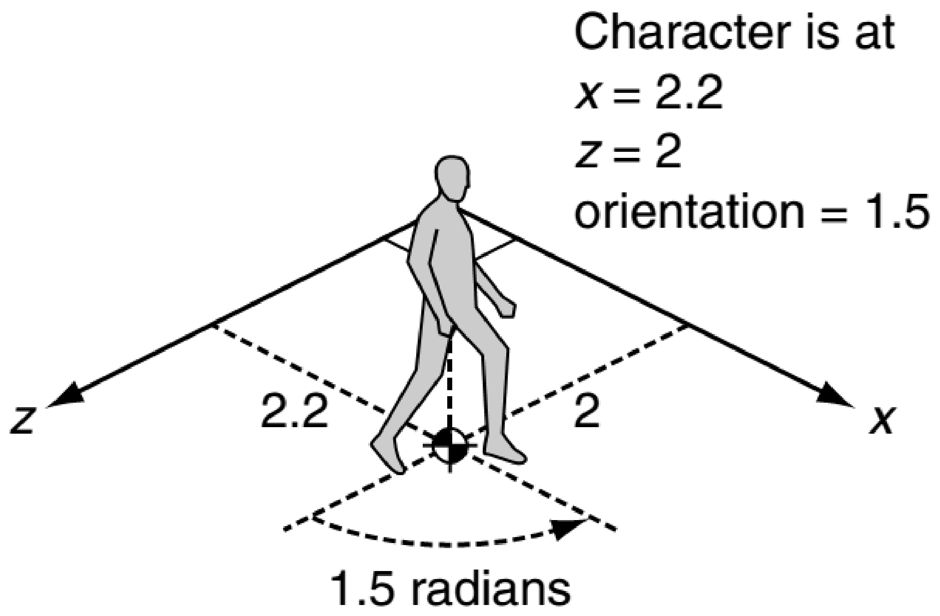
Millington Fig 3.2

Assumptions

- Computed quickly
- Impression of intelligence (&reality), not a simulation
- Character position model: point + orientation
- Full 3D usually unnecessary (ie scalar Θ)
  - 2D suffices, thanks to gravity
    - (x, y, Θ) … 3 degrees of freedom
  - 2½ D (3D position, 2D orientation) covers most
    - (x, y, z, Θ) … 4 degrees of freedom
- *Rotation* is the process of changing *orientation*

Space

- Axes
- Orientation
- Local vs global coordinate systems

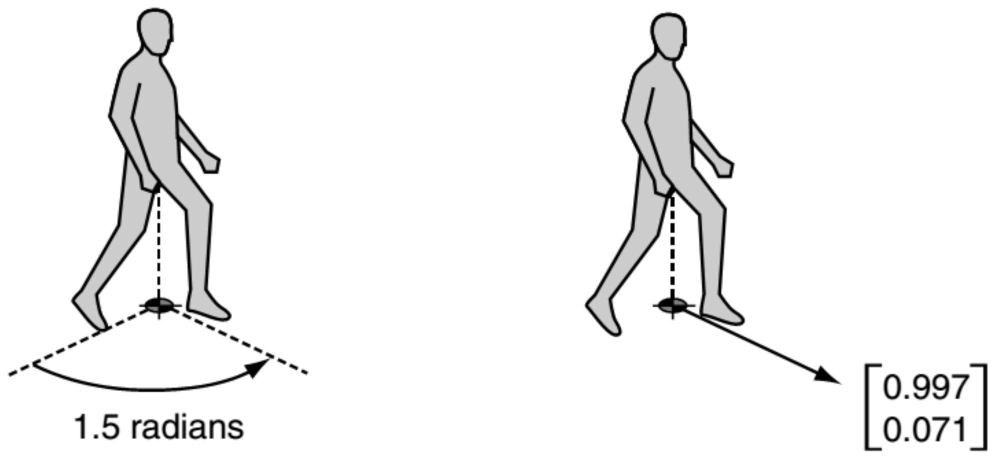


Millington Fig 3.4

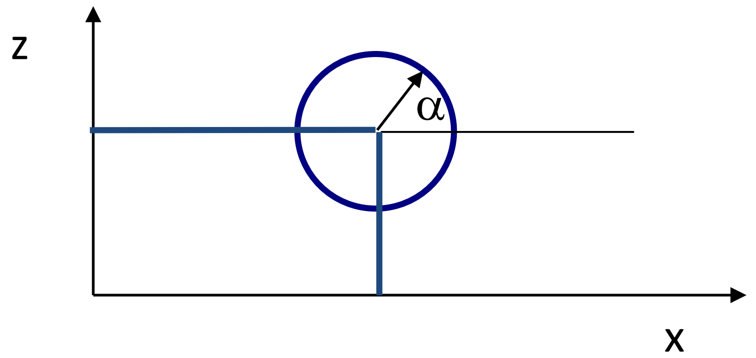
Vector Form of Orientation

- Convenient to represent orientation as unit vector (len = 1)

- $\overrightarrow{\omega_v} = [\sin \alpha_s, \cos \alpha_s]$



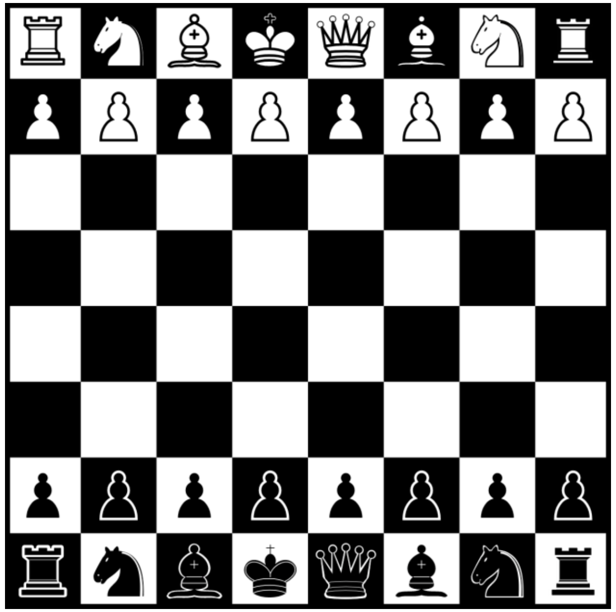
Millington Fig 3.5



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

Statics

- Static, because no information about movement
  - Position
    - 2 or 3-dimensional vector
  - Orientation
    - 2-dimensional unit vector given by an angle (e.g. [0.997, 0.071]) OR a single real value between 0 and 2 π (e.g. 1.5)
- What do movement algorithms output?



```
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 2π
  - **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 2π
- 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
```

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

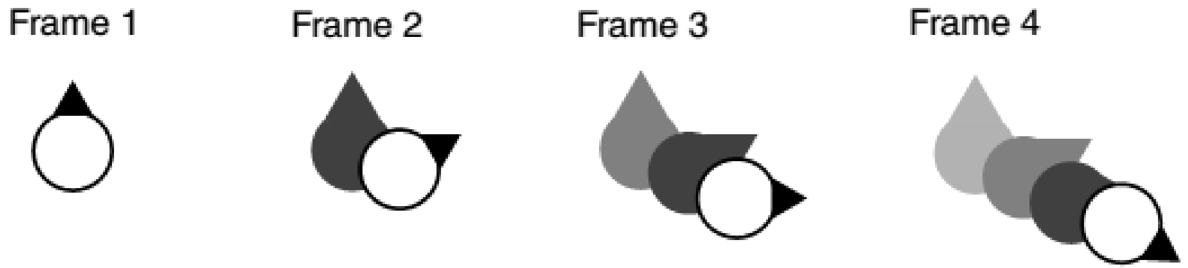
Sidebar: Time & Variable Frame Rates

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

FACING?

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: change orientation to be halfway toward current direction of motion in each frame



Millington Fig 3.6

Changing Orientation (facing)

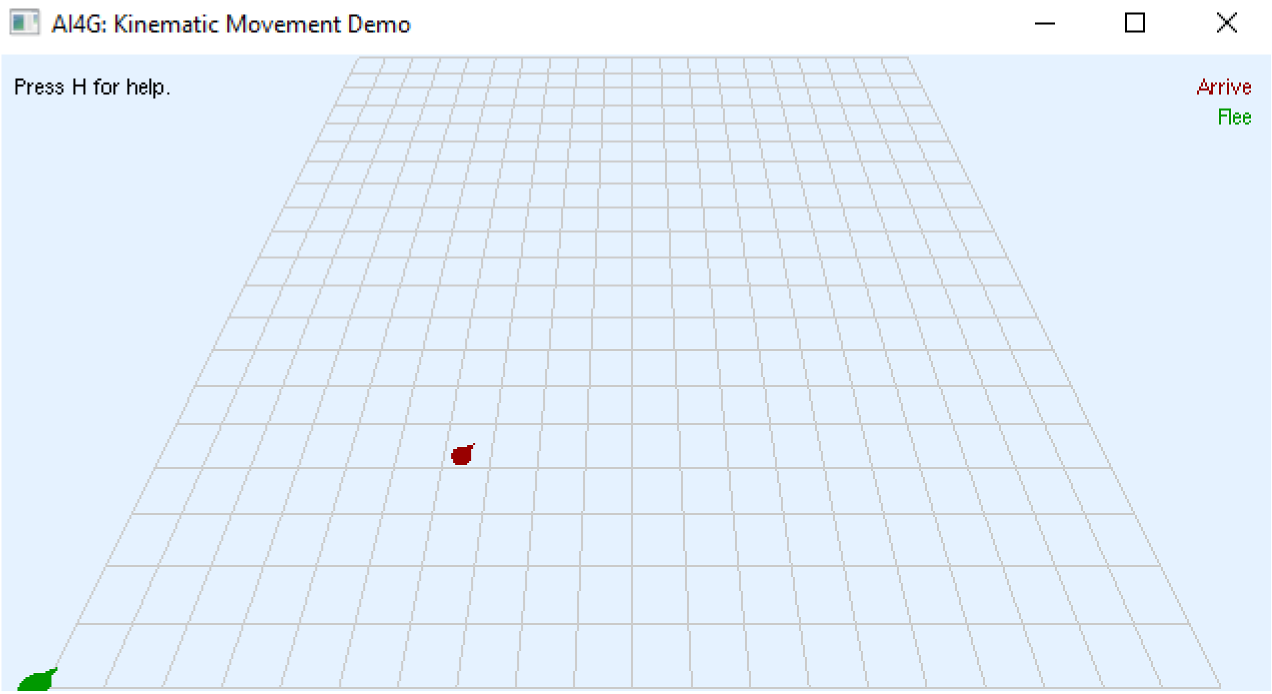
- Uses static data (position & orientation)
- Outputs desired velocity
  - On/off in target direction
  - Smoothing may be done (without *a*)
- New *v* determines new Θ
  getNewOrientation(currentOrientation, targetVelocity)
  - If v > 0, return interpolation between current and desired orientation
    [atan2(-static.x, static.z)]
  - Else use current orientation

But not necessarily in that order

SEEK, ARRIVE, FLEE, AND WANDER?

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

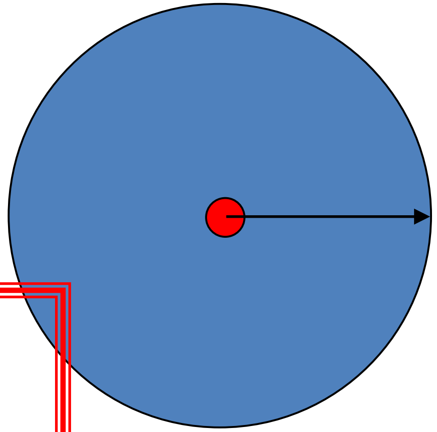

AI4G: Kinematic Movement Demo
Press H for help.
Arrive
Flee

Kinematic Arrival

- Seek with full velocity leads to overshooting
  - Arrival modification?

Kinematic Arrival

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



```
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();
```

Arrival Circle:

Slow down if you get here

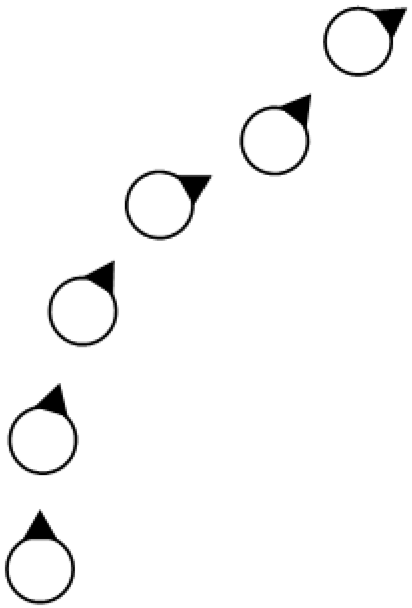
Millington 3.2.1

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

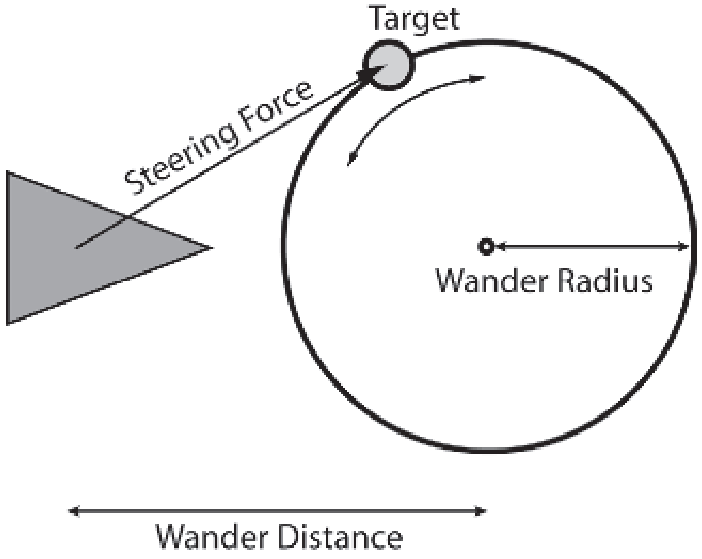

Steering Behaviors - Arrive
MaxForce(Ins/Del): 2.00
MaxSpeed(Home/End): 150.00
Click to move crosshair

Kinematic Wander

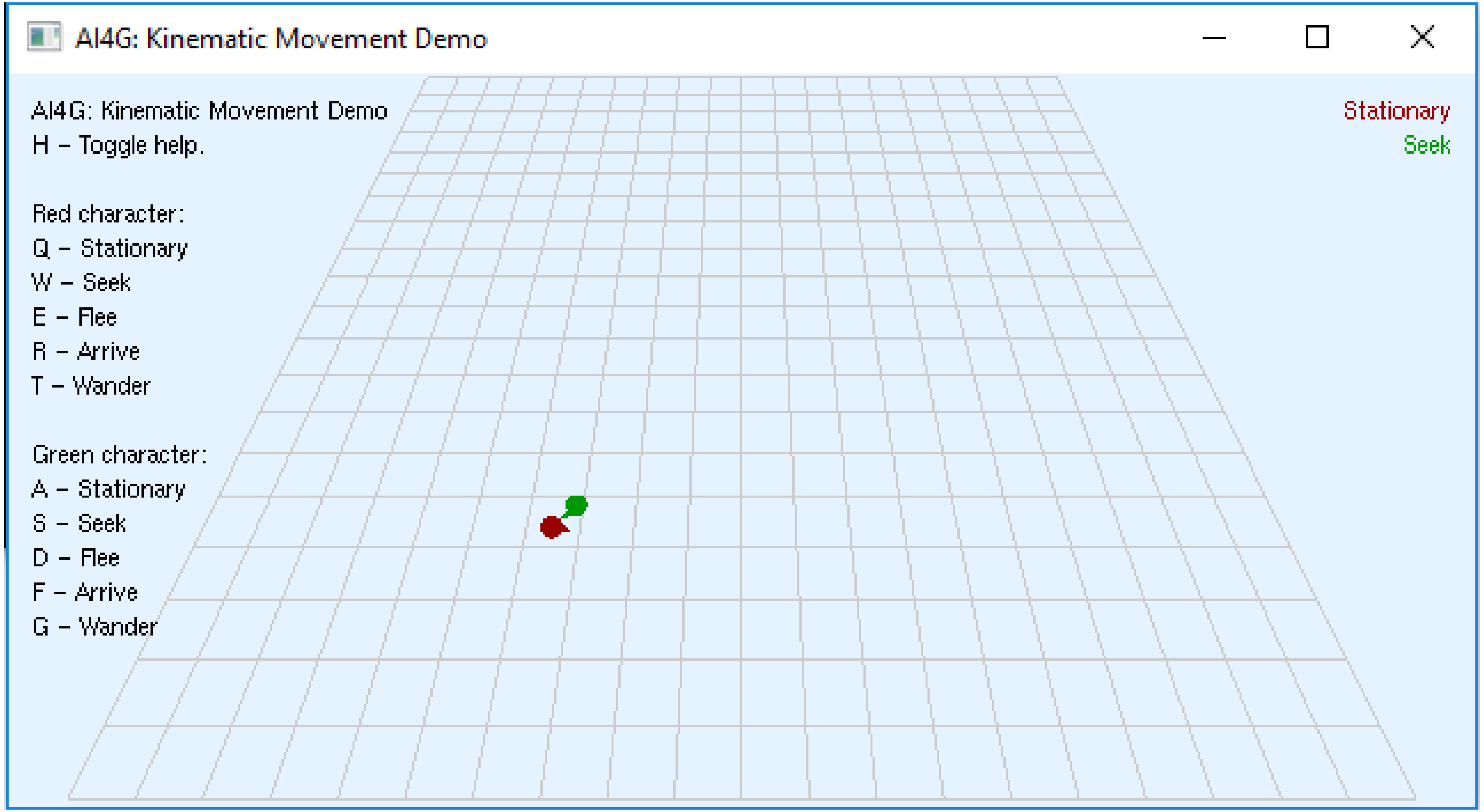
- Move in current direction at max speed
- Vary orientation by some random amount each frame



Millington Fig 3.7



Buckland Fig 3.4


AI4G: Kinematic Movement Demo
AI4G: Kinematic Movement Demo
H – Toggle help.
Red character:
Q – Stationary
W – Seek
E – Flee
R – Arrive
T – Wander
Green character:
A – Stationary
S – Seek
D – Flee
F – Arrive
G – Wander
Stationary
Seek

Kinematics

- Computing a new target velocity based on {x,z} + Θ can look unrealistic
  - Can lead to abrupt changes of velocity
  - Must smooth velocity (or use acceleration model)
- {x,z} + Θ + v → can increment velocity by some Δ from $curr_v$ up to $target_v$
- Must track velocity in all dimensions plus rotation

Kinematic 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
    - 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

Kinematic 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

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)
  - Download sample materials: http://www.jblearning.com/catalog/9781556220784/
- Animations (for simple). Craig Reynolds
  - http://www.red3d.com/cwr/steer/
- http://en.wikipedia.org/wiki/Radian

Steering Behaviors (Dynamic)

- Kinematic movement
  - Outputs: desired velocity
- Steering movement (behaviors)
  - Input: target information
    - Velocity and rotation
    - Collision geometry
    - Paths, for path following
    - Average Flock information
  - Output: accelerations
    - Linear acceleration: 2 or 3-D vector
    - Angular acceleration: single float value

- Steering extends kinematic movement by **adding acceleration and rotation**
  - Remember:
    - $\mathbf{p}(t)$: position at time t
    - $\mathbf{v}(t) = \mathbf{p}'(t)$: velocity at time t
    - $\mathbf{a}(t) = \mathbf{v}'(t)$: acceleration at time t
  - Hence:
    - $\Delta\mathbf{p} \approx \mathbf{v}$
    - $\Delta\mathbf{v} \approx \mathbf{a}$

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 Movement

- 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

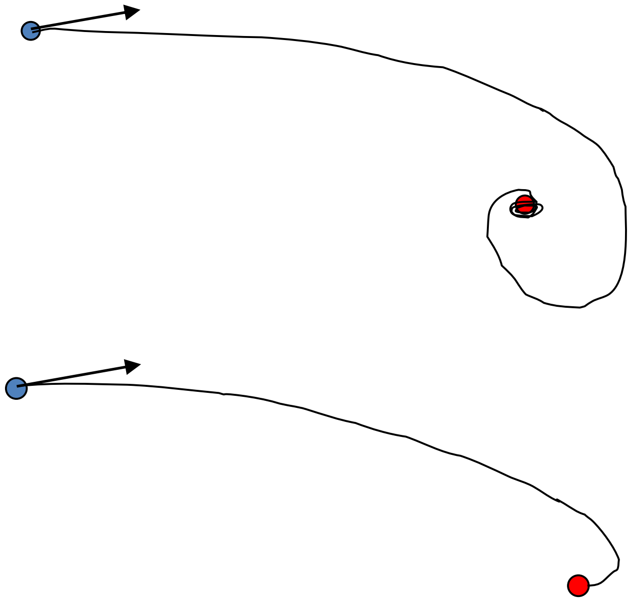
Seek + Arrive

Variable Matching

- Simplest family: match one or more elements of source to target
  - Match **position** (seek/flee): 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

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, https://github.com/idmillington/aicore

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

Other behaviors?

- Pursuit / Evade
- Hide
- Obstacle & Wall Avoidance
- Path following (list of points)

- Groups? E.g. offset pursuit

Derived & Composite Steering Behaviors

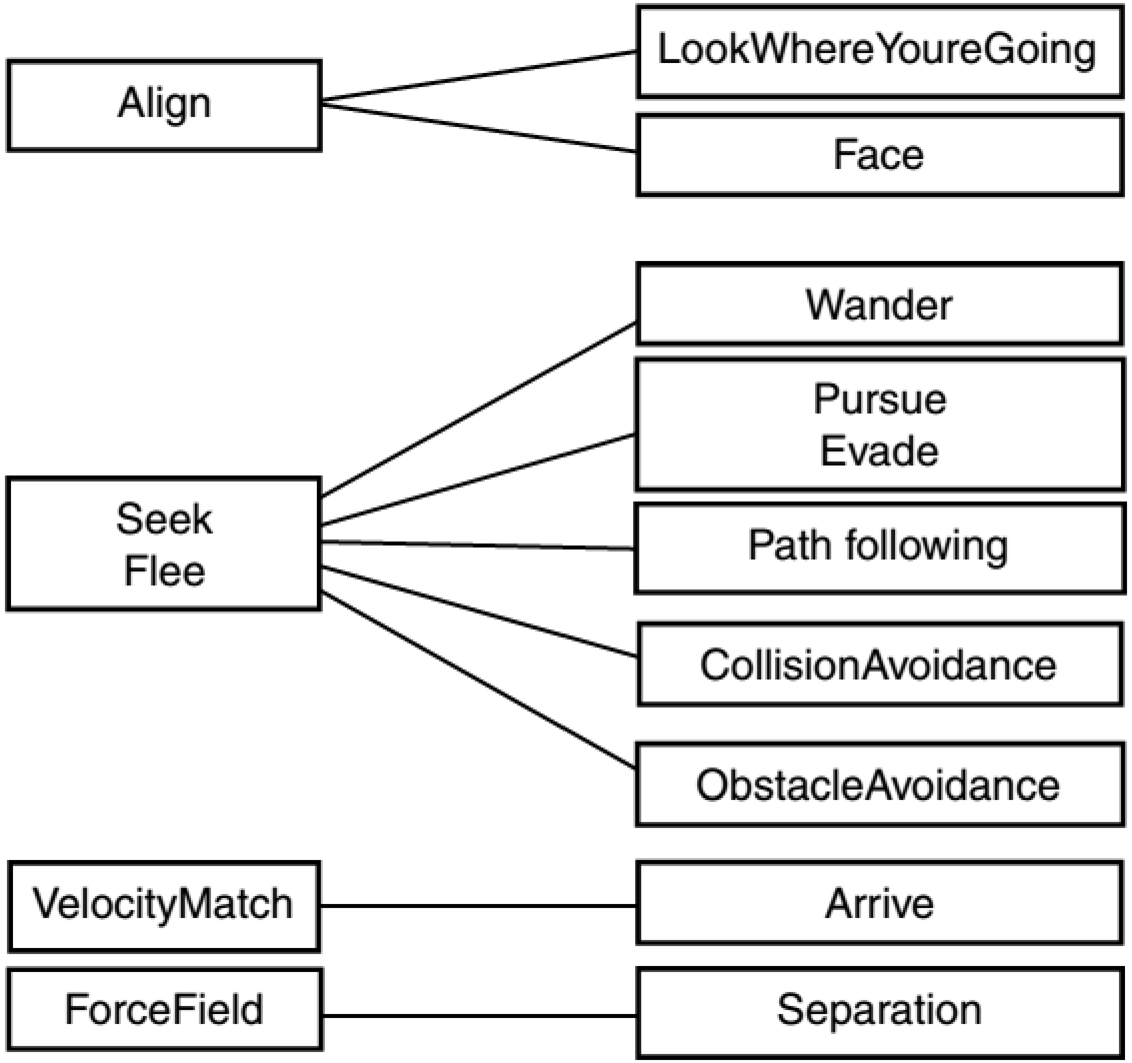
- More complex behaviors derived from core
  - Pursue (evade): Seek (flee) based on predicted target position
  - Face: Align to target orientation
  - Look where going: Face in direction of movement (using Align)
  - Collision avoidance: Flee based on obstacle proximity
  - Wander: Seek + Face some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

Demo

- Pursuit
- Obstacle Avoidance

Composite Behaviors

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



Millington Fig 3.29

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