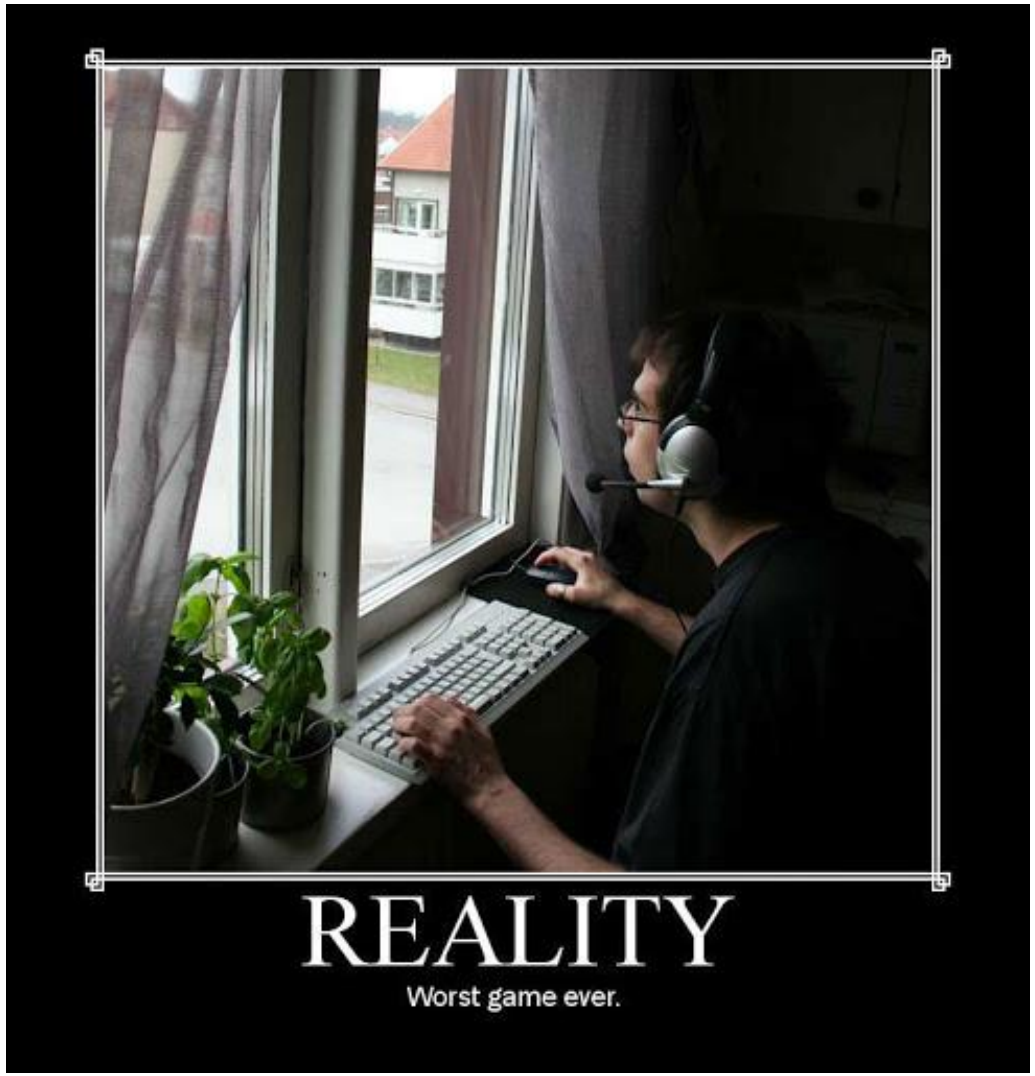
Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

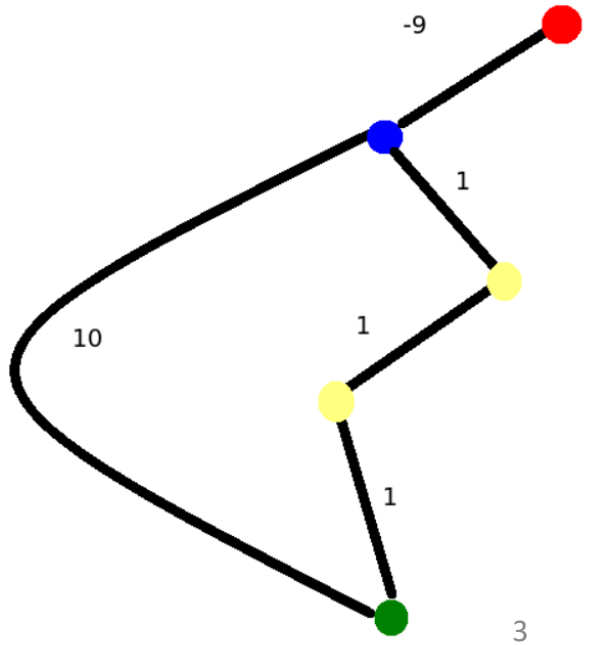


Announcements

- Start HW3!
- Everyone is verified
- Emails (please use ‘GTGameAI’ as first word of subject)
- Piazza poll 2: digital distractions
- Math refresher: Buckland ch 1
  - Alternate obstacles discussion: Buckland pg 99 (pdf 122/521)
- Webpage: https://www.cc.gatech.edu/~surban6/2018sp-gameAI/
  - Readings posted for next few classes

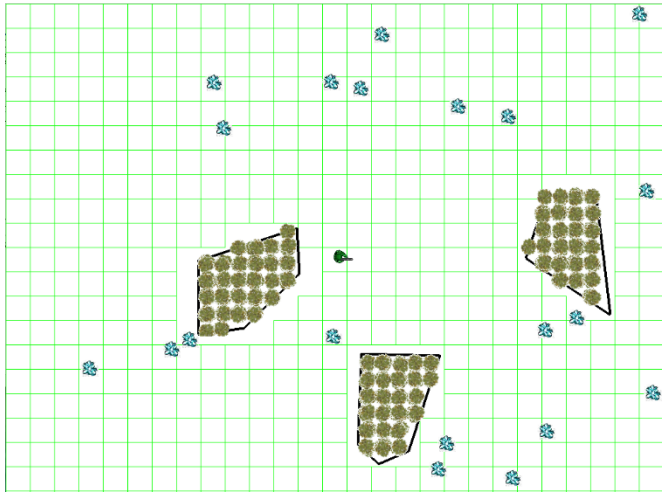
Erratum

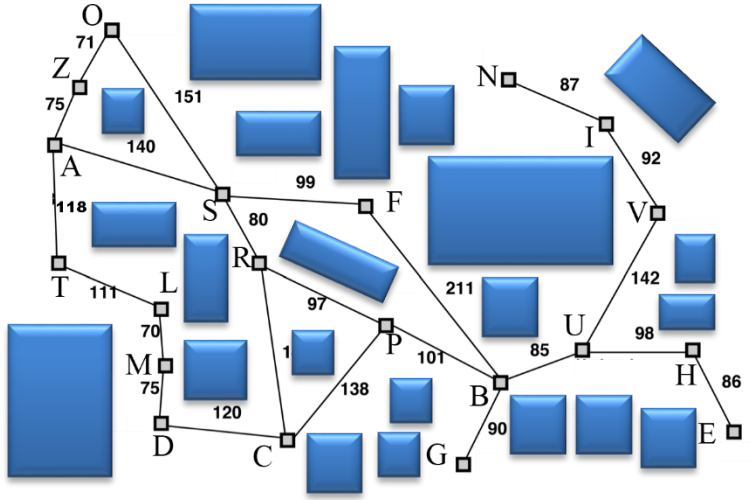
- Making graphs non-negative: Increasing by absolute value of minimum changes shortest path (credit: Thor). Don't do that.
  - it is possible to reweight a network (with no negative cycles) in a shortest-path preserving way
  - https://piazza.com/class/jc71bf1wrdg6d6?cid=123
- D* lite in games (credit: Andrew)
  - May or may not be impractical due to memory use
  - https://piazza.com/class/jc71bf1wrdg6d6?cid=124

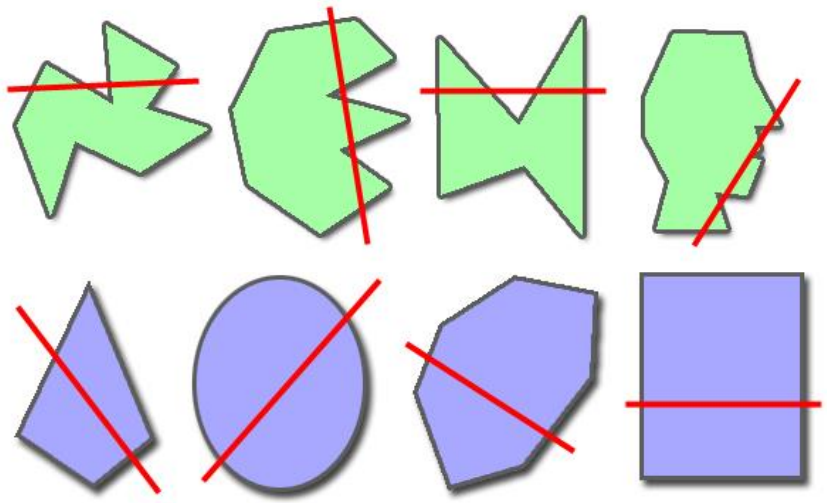


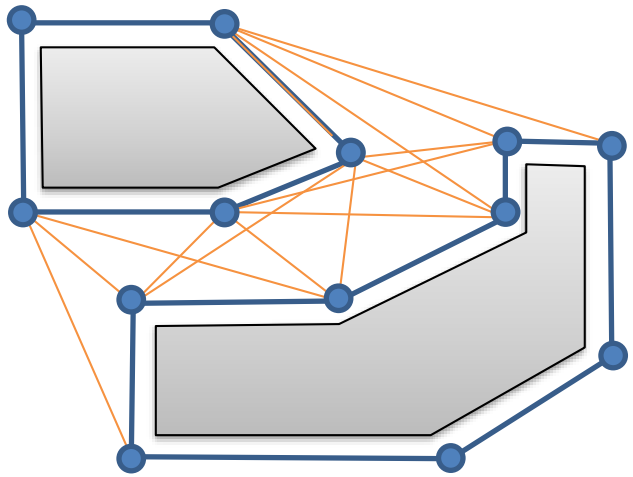
PREVIOUSLY ON…

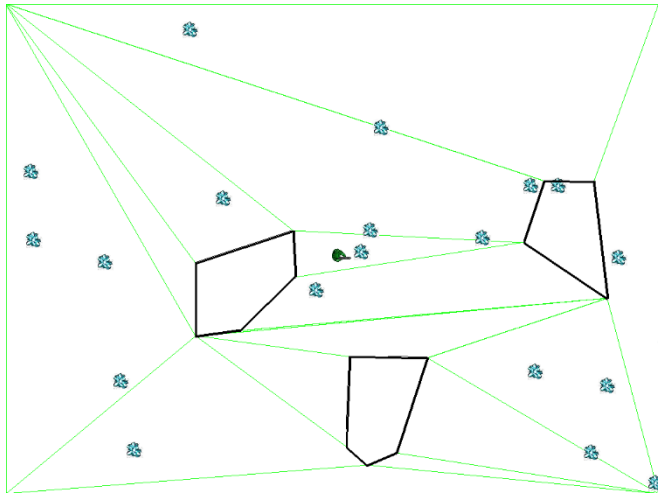
Modelling and Navigating the Game World

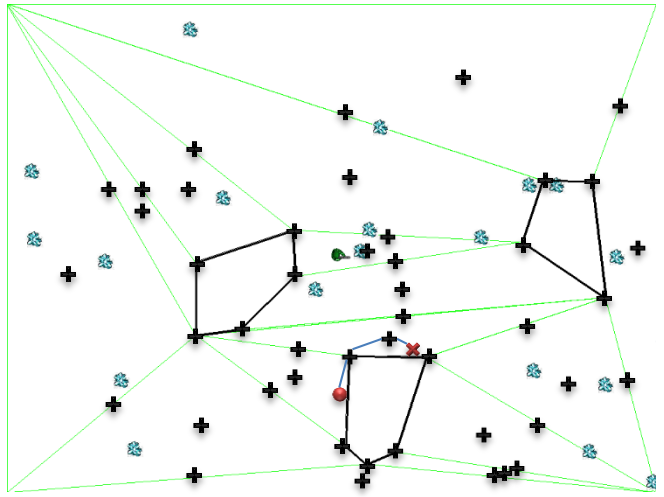




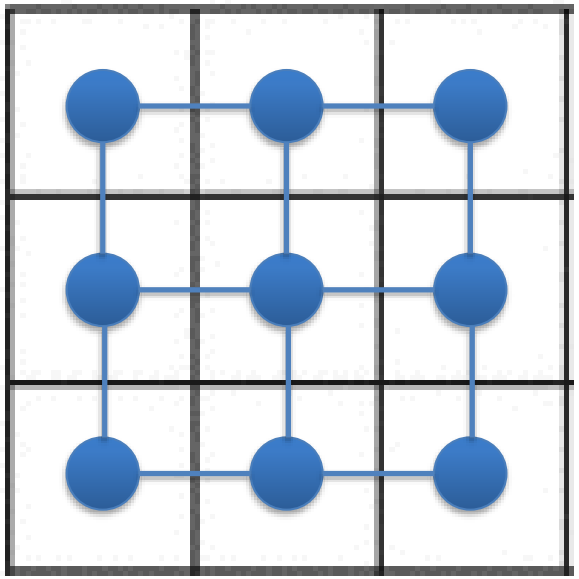


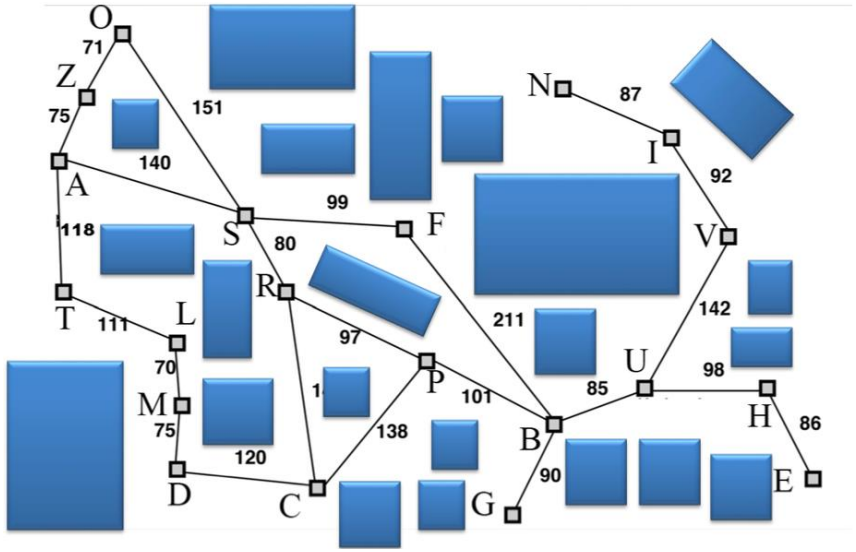


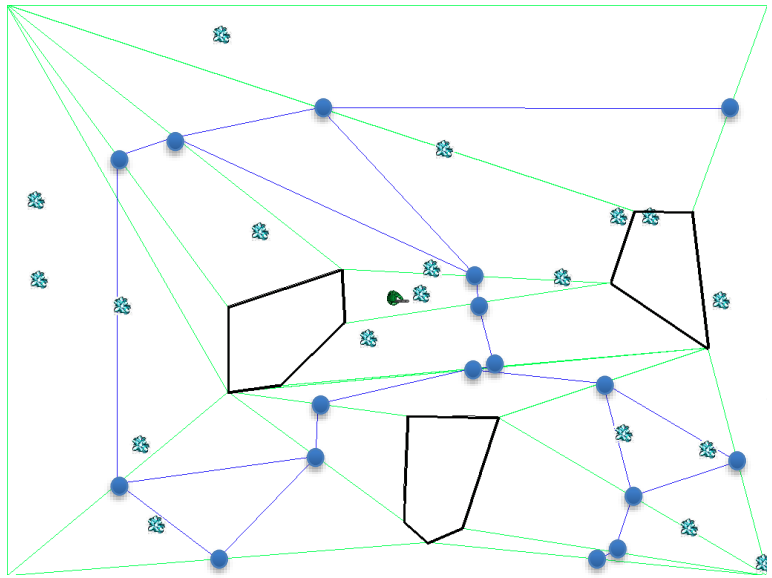




Graphs, Graphs, Graphs…







Graph Search: Sorting Successors

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

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

N-2

1. What kind of solution does greedy search find? Why might this be useful?
2. What kind of solution does A* find?
3. What are some of the insights behind A*?
4. What's a good data structure to use with A*? Why?

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?

N-1: Heuristic Search Recap

- A*
  - Use when we can't precompute
    - Dynamic environments
    - Memory issues
  - Optimal when heuristic is admissible (and assuming no changes)
  - Replanning can be slow on really big maps
- Hierarchical A* is the ~same, but faster
  - Within 1% of A* optimality but up to 10x faster
- Real-time A*
  - Stumbling in the dark, 1 step lookahead
  - Replan every step, but fast!
  - Realistic? For a blind agent that knows nothing
  - Optimal when completely blind
- Real-time A* with lookahead
  - Good for fog-of-war
  - Replan every step, with fast bounded lookahead to edge of known space
  - Optimality depends on lookahead

N-1: Heuristic Search Recap

- D* Lite
  - Assume everything is open/clear
  - Replan when necessary
  - **Worst case: Runs like Real-Time A***
  - **Best case: Never replans**
  - Optimal including changes

Graphs, Search, Pathfinding

(behavior involving **where** to go)

Steering, Flocking, Formations

(behavior involving **how** to go)

(kinematic) Movement, Steering, Flocking, Formations

2018-01-30

M&F 3.1-3.4

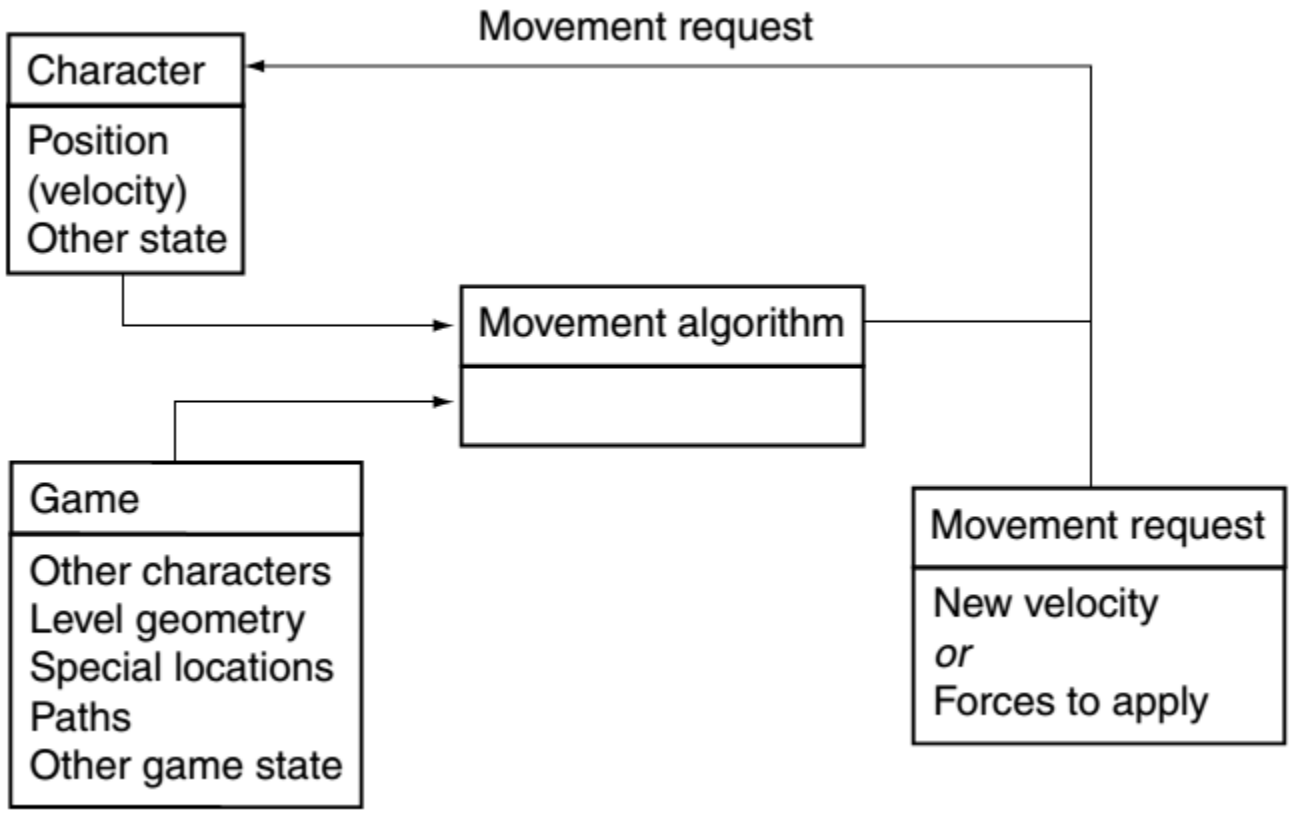
B 3

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 $\subset$ steering

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

General Algorithm



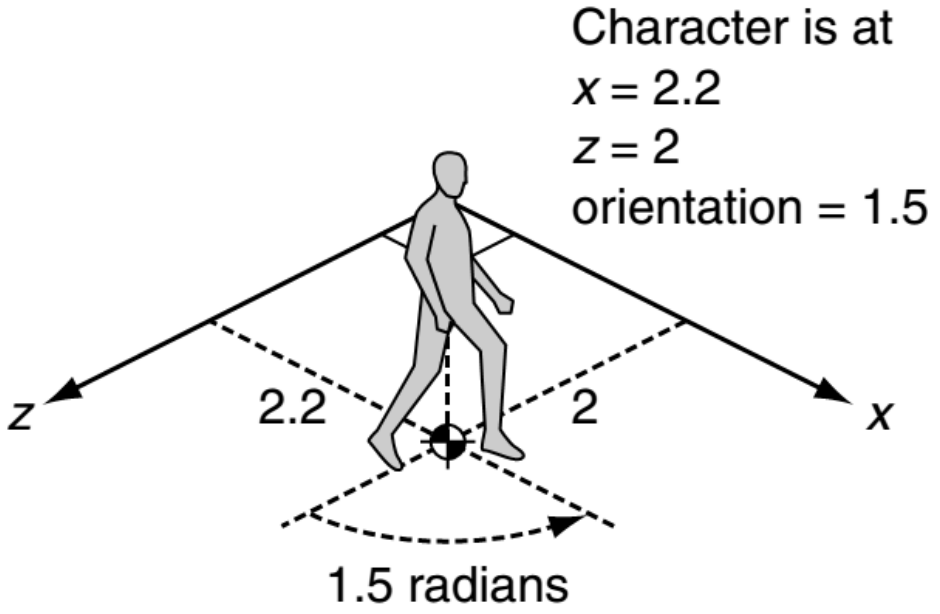
Millington Fig 3.2

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

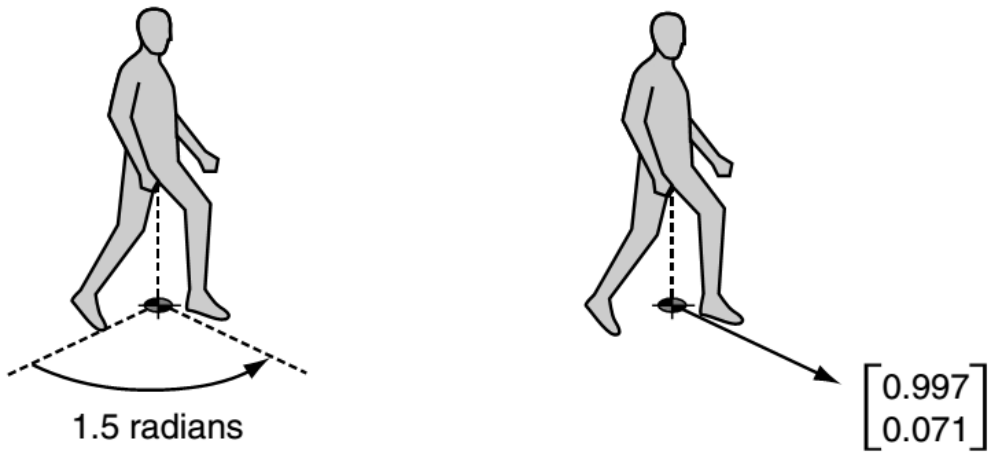
- Axes
- Orientation
- Local vs global coordinate systems



Millington Fig 3.4

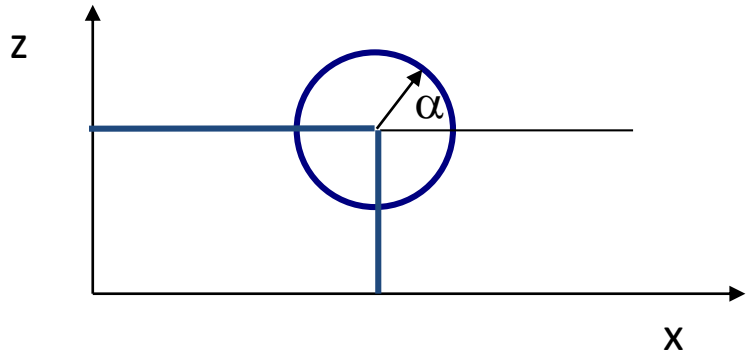
Vector Form of Orientation

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



Millington Fig 3.5

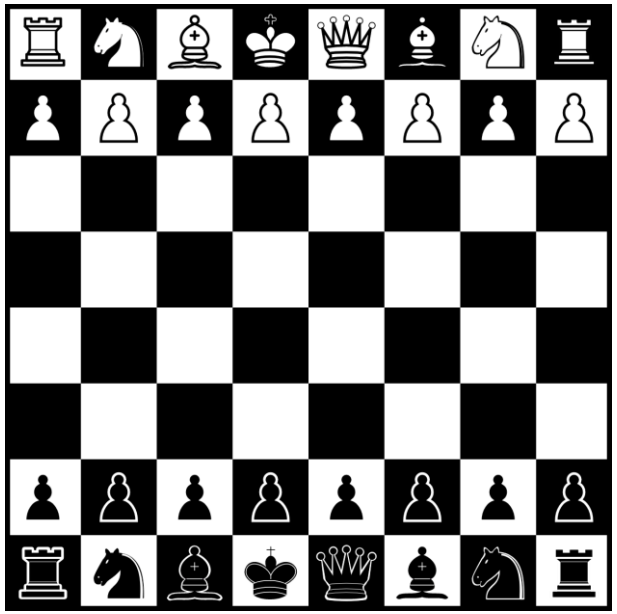
- $\overrightarrow{\omega_v} = [\sin \alpha_s, \cos \alpha_s]$



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?



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 π
  - **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 π
- Movement behaviors output
  - Velocity
  - Rotation

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

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

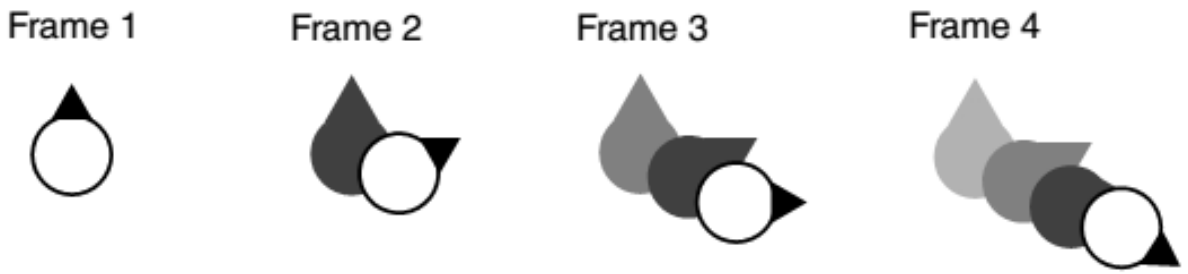
- steering.linear: a 2D vector
  - Represents changes in velocity (linear acceleration)
- steering.angular: a real value
  - Represents changes in orientation (angular acceleration)

- 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

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

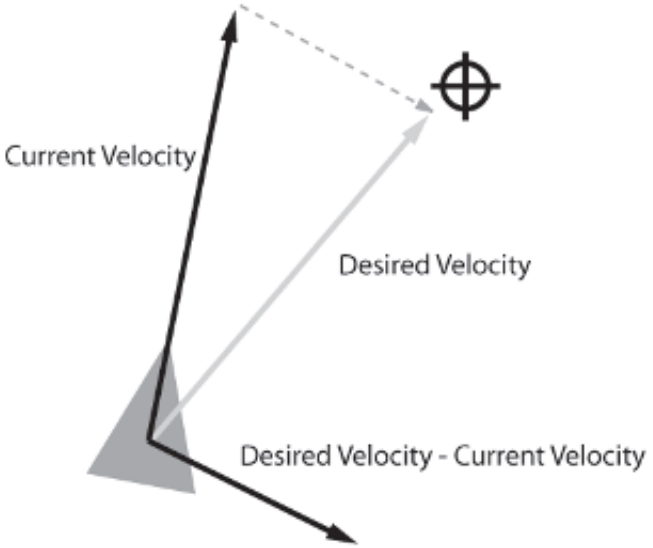
Changing Orientation (facing)

- 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



B Fig 3.2

- directs an agent toward a target position
- 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 to face (prev slide)
- O(1) in time and memory
- Flee = -1 * velocity = character.position – target.position


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

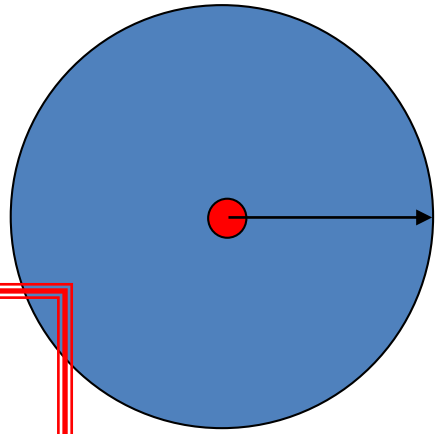

Steering Behaviors - Flee
Click to move crosshair
Press R to reset

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



Arrival Circle:

Slow down if you get here

```
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();
```

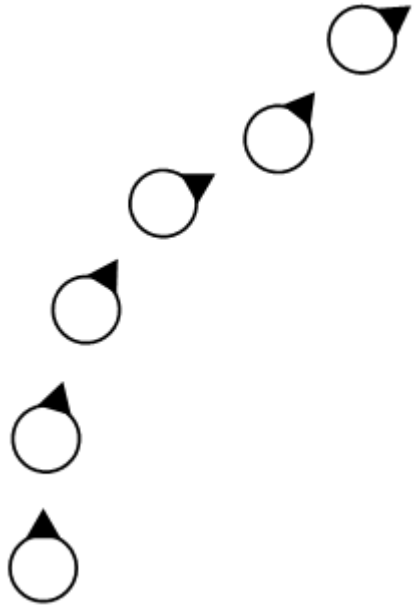
Millington 3.2.1

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

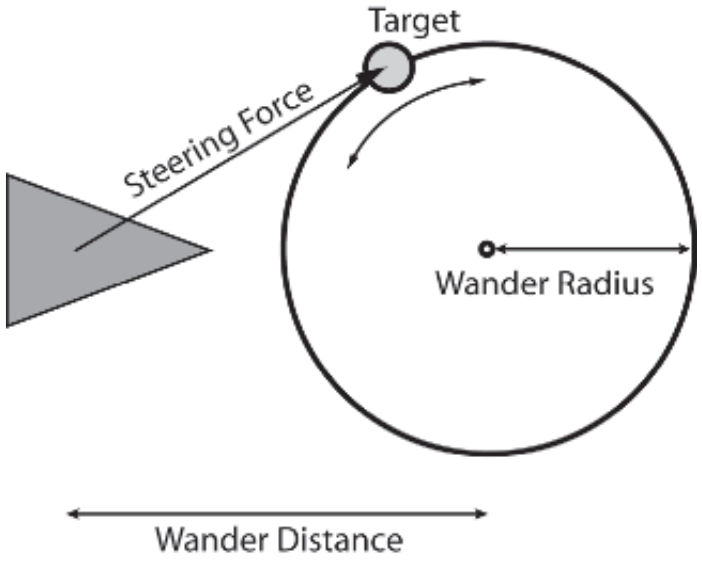

Steering Behaviors - Arrive
MaxForce(Ins/Del): 2.00
MaxSpeed(Home/End): 150.00
Click to move crosshair

Kinematic Wander

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



Millington Fig 3.7

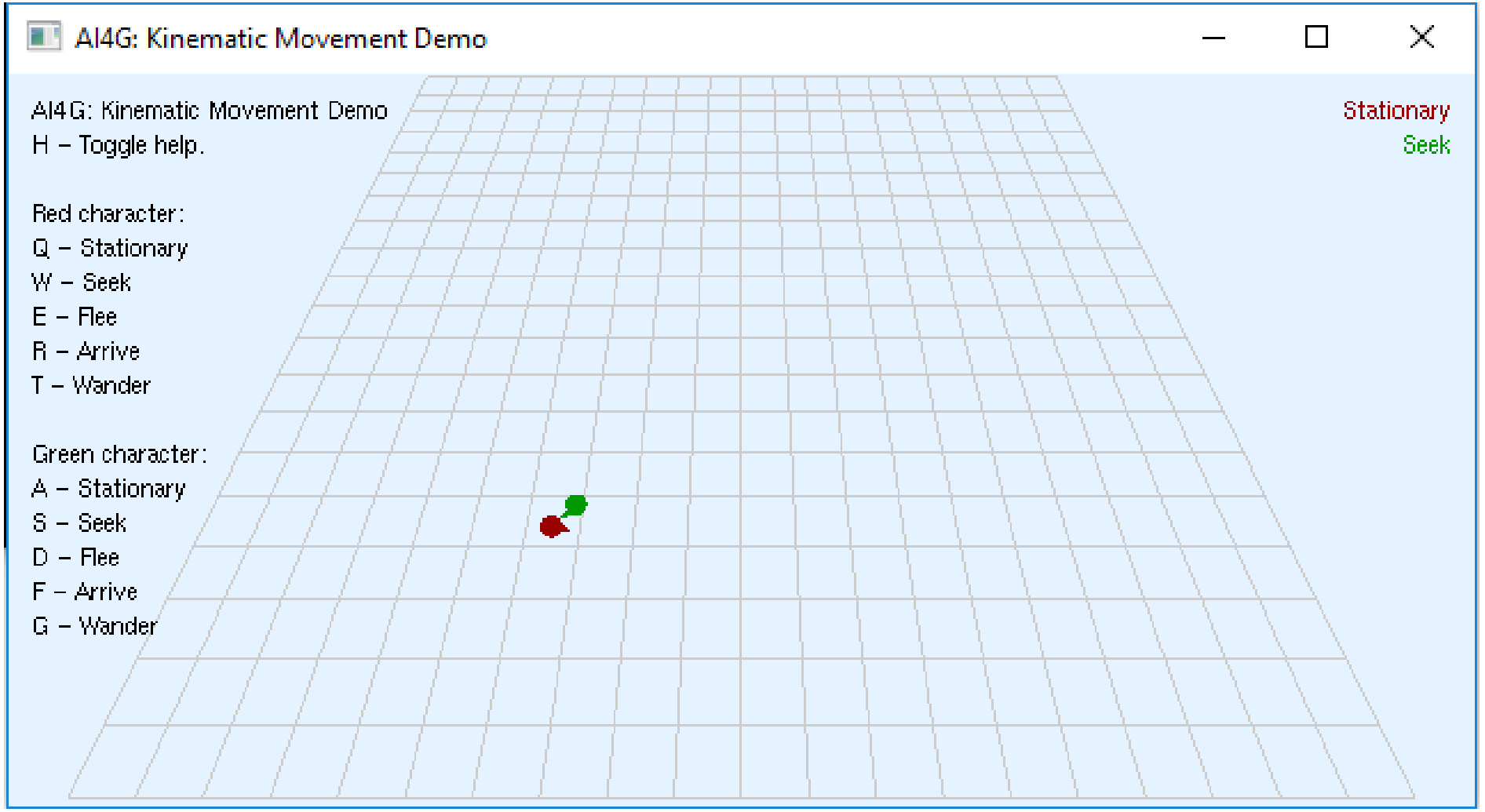


Buckland Fig 3.4

Other behaviors?

- Pursuit / Evade
- Hide
- Obstacle & Wall Avoidance
- Path following

- Groups? E.g. offset pursuit


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

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