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

- HW2 discussion
- Start HW3!
- 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)

PREVIOUSLY ON…

N-3

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-2: 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-2: 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: Movement 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*

N-1: Movement

1. What do movement algorithms output in static environ?
2. What do movement algorithms output in kinematic environ?
3. What do movement algorithms input 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?

Graphs, Search, Pathfinding

(behavior involving **where** to go)

Steering, Flocking, Formations

(behavior involving **how** to go)

Dynamic Movement, Steering

2018-01-30

M&F 3.3

B 3

See also

- M Book, Ch 3.1-3.4. Also 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/

Steering Behaviors (Dynamic)

- Kinematic movement
  - Input: use target position and orientation, no velocities.
  - 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

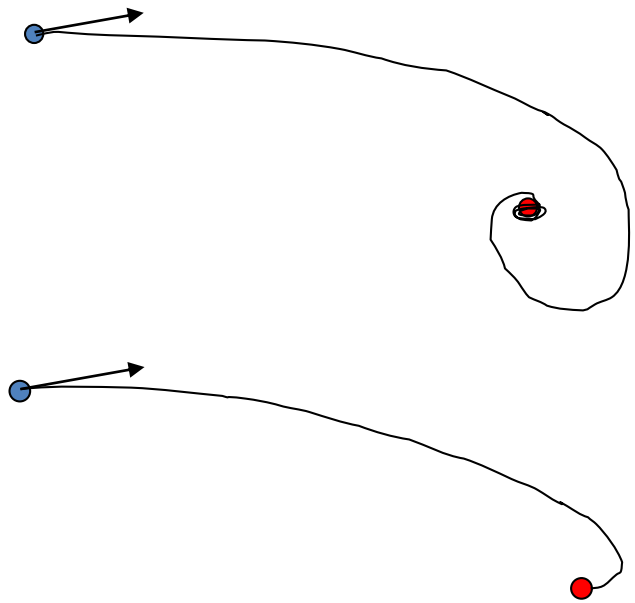
- Dynamic movement update
  - Accelerate in direction of target until maximum velocity is reached
  - (Optional) If target is close, lower velocity (Braking)
    - Negative acceleration is also limited
  - (Optional) If target is very close, stop moving

Seek + Arrive

- 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

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

Derived & Composite Steering Behaviors

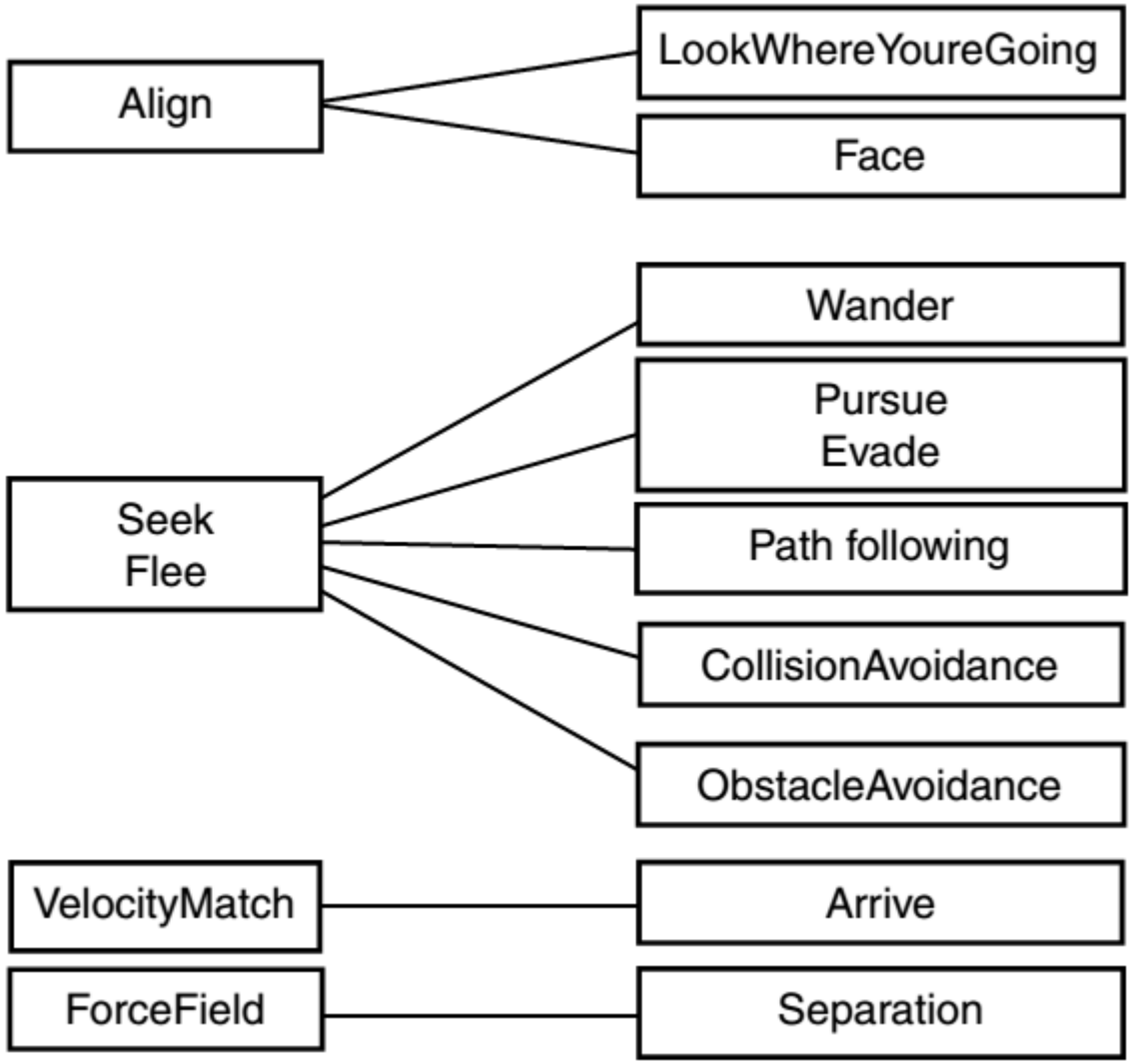
- More complex behaviors derived from core
  - Pursue (evade): Seek (flee) based on predicted target position
  - Face: Align to target orientation
  - Look where going: Face in direction of movement (using Align)
  - Collision avoidance: Flee based on obstacle proximity
  - Wander: Seek + Face some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

Demo

- Pursuit
- Obstacle Avoidance

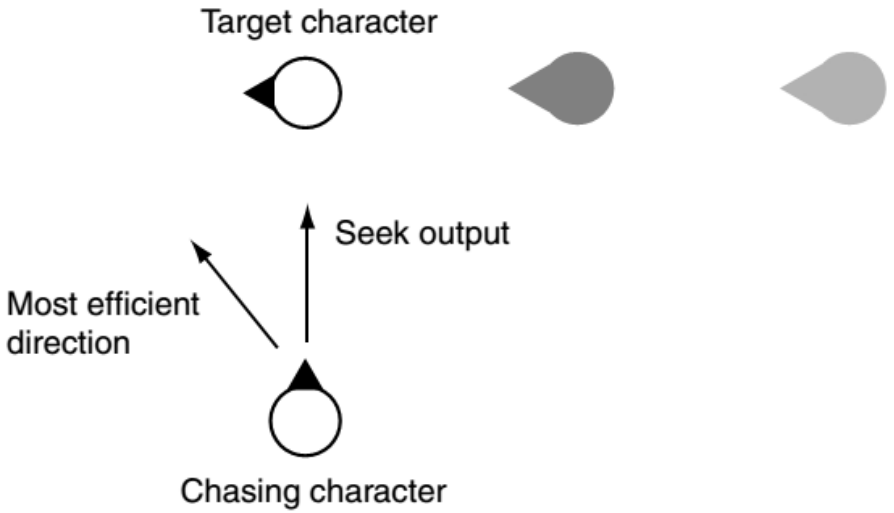
Composite Behaviors

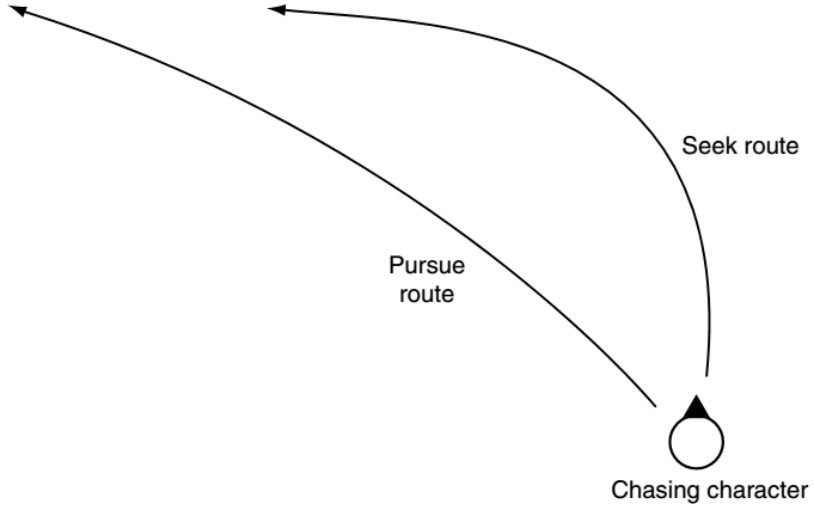
- Pursue / Evade
- Face / Look where going
- Wander
- Collision Avoidance
- Obstacle Avoidance
- Separation



Millington Fig 3.29

Composite Behaviors

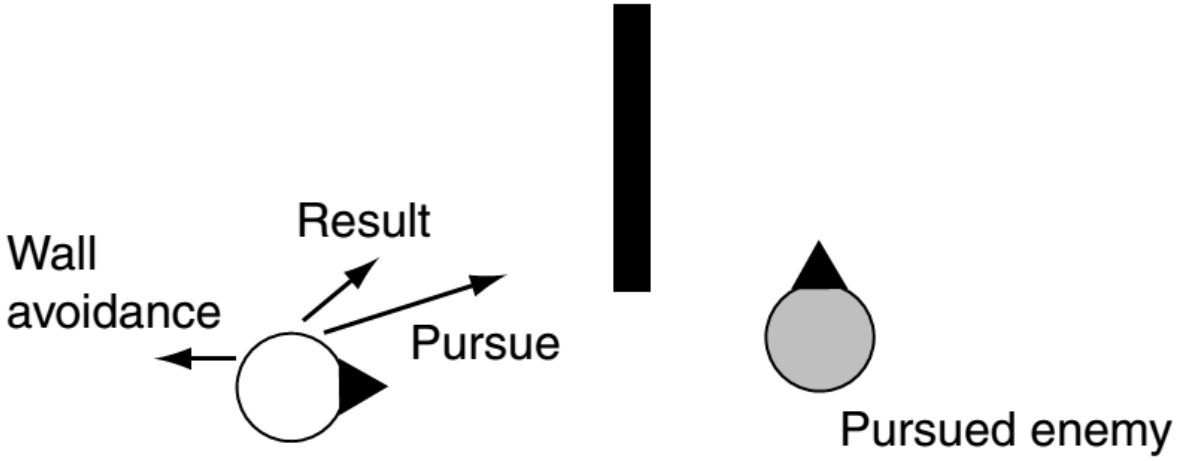


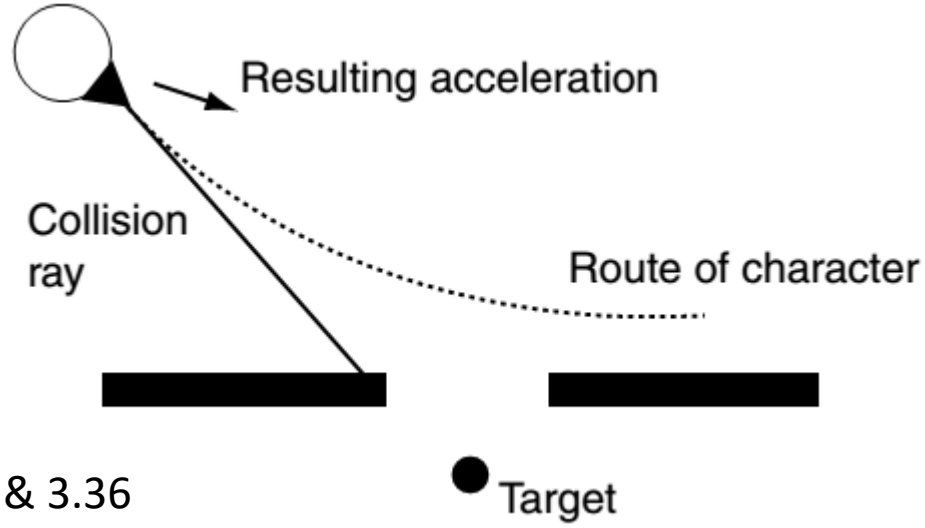


Millington Fig 3.11 & 3.12

Variable Matching Conflicts

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





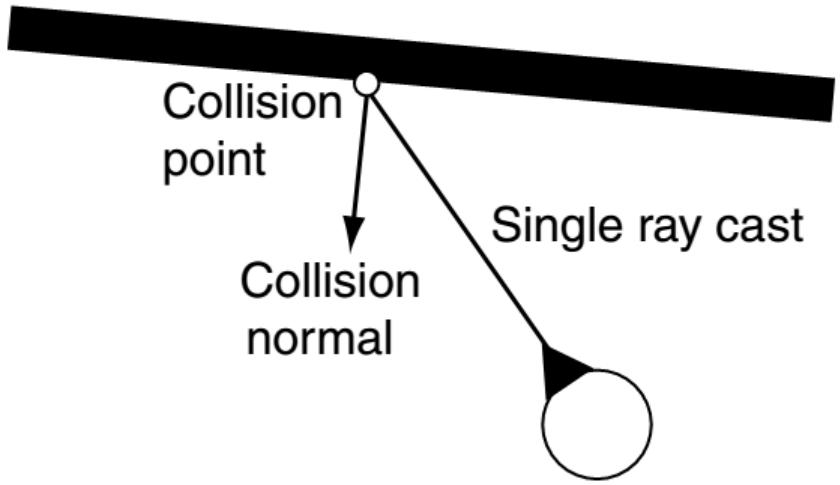
Millington Fig 3.35 & 3.36

can't avoid an obstacle and chase

missing a narrow doorway

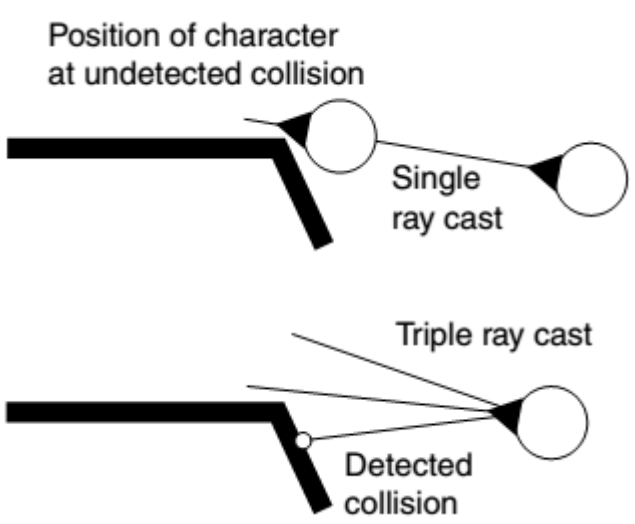
Obstacle and Wall Avoidance

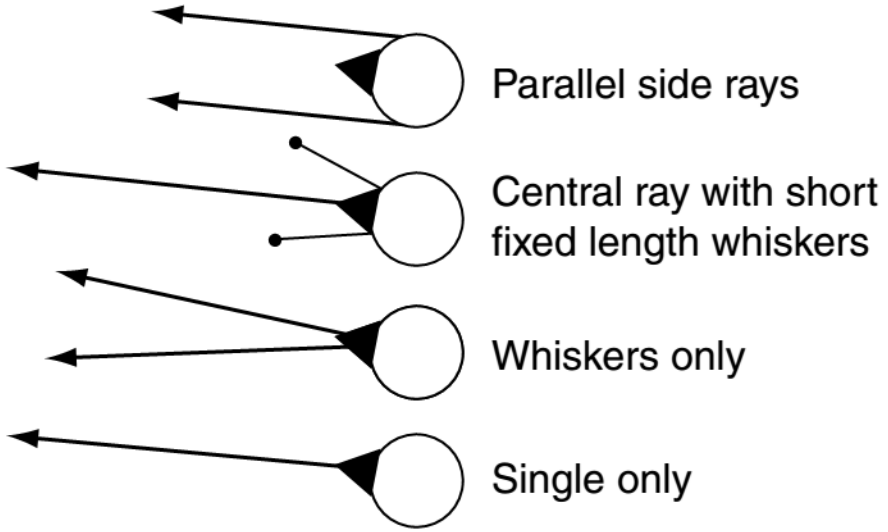
- Cast one or more (distance-bounded) rays out in direction of motion
- Use collisions to create sub-target for avoidance
- Perform basic seek on sub-target



Millington Fig 3.24

One is not enough

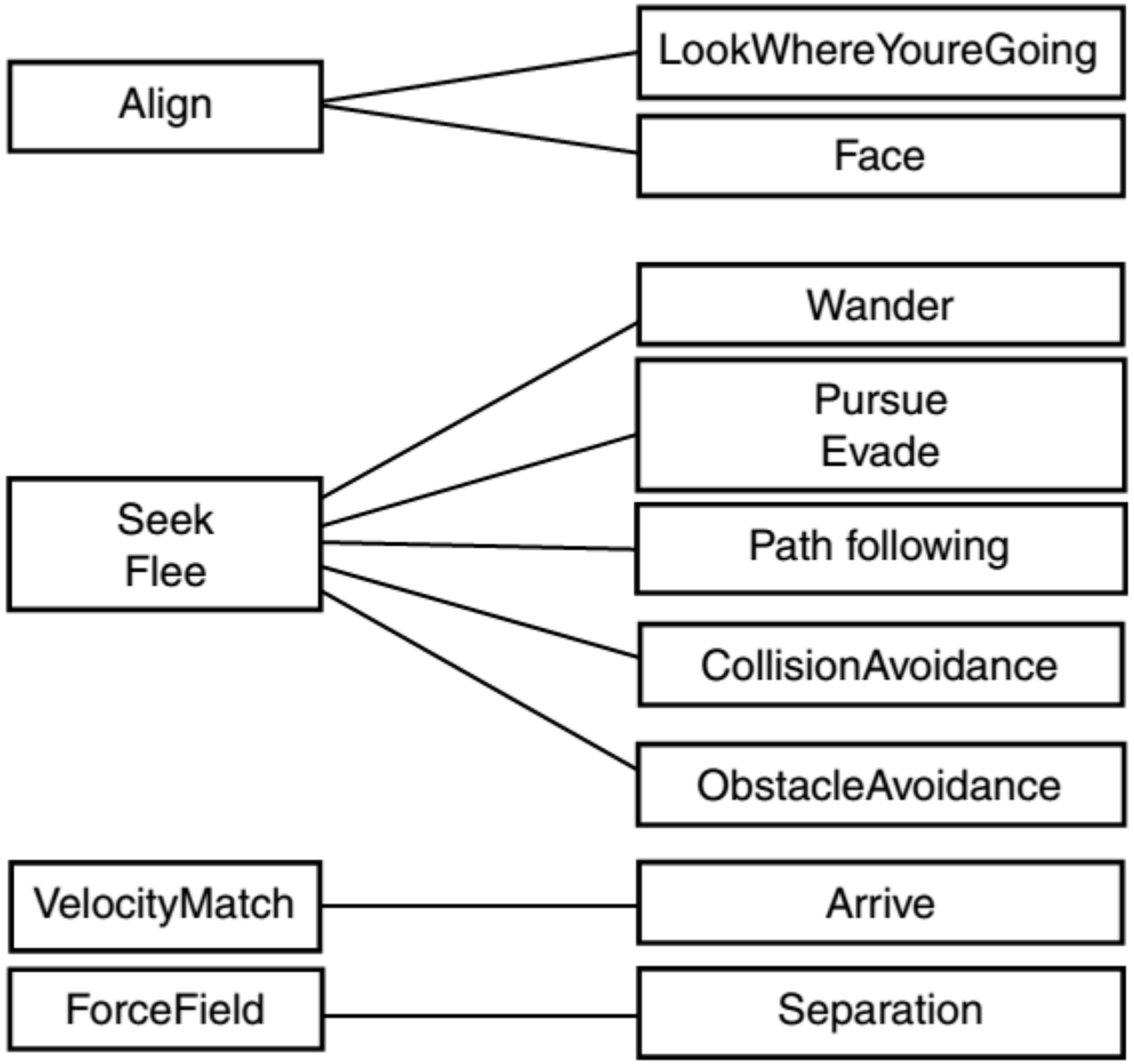




Millington Fig 3.25 & 3.26

Composite Behaviors

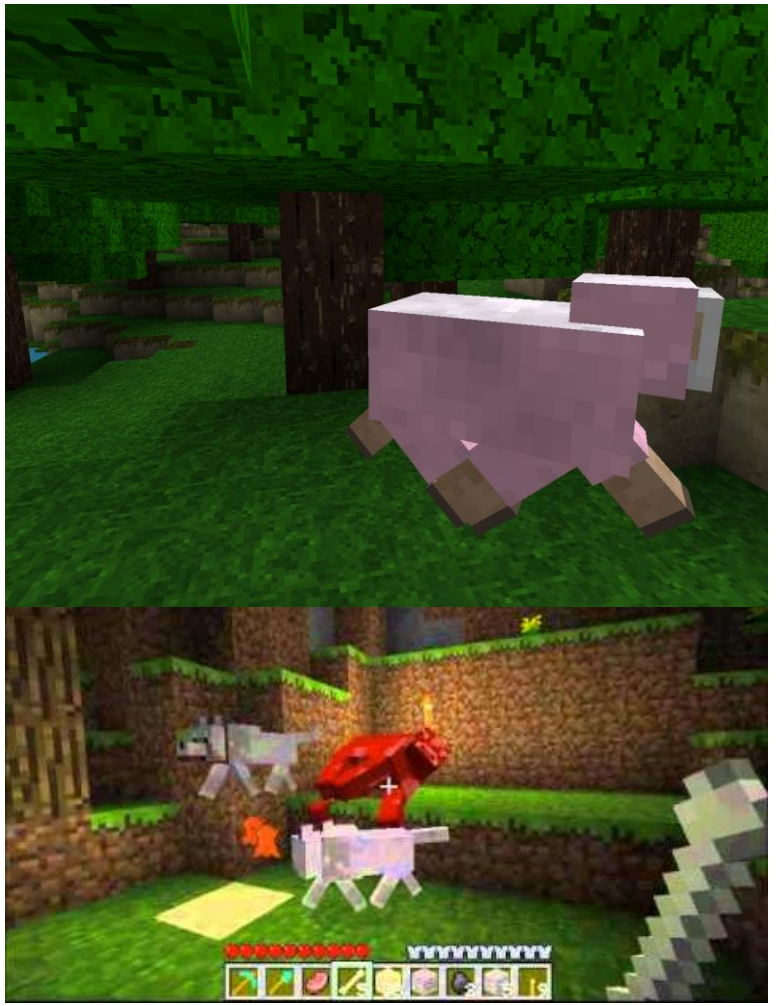
- Pursue / Evade
- Face / Look where going
- Wander
- Collision Avoidance
- Obstacle Avoidance
- Separation



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
- 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*sep + 1*cohes
- Finding “right” weight can be challenging
  - Characters can get stuck (equilibrium)
  - Constrained environments (conflicts)
  - Jidder

Demo

- Blending priority