Game AI

Overview

- AI in games
- Movement
  - Movement algorithms
  - Steering behaviours
  - Group behaviour
- Controlling behaviour
  - State machines

Game AI

- Artificial intelligence in games
  - The purpose is to entertain and challenge the player
    - Not to simulate the human mind
    - Create the illusion of intelligence
  - Player expectations about AI
    - Should not fail obvious tasks
    - Assist with storytelling and create living worlds
  - Difficult to develop AI that can perform well in all situations
    - Typically synchronised to the level features and geometry
    - Additional computational complexity for AI that can handle multiple situations

Game AI

- To compensate for AI shortcomings
  - Challenge the player with the number of AI entities
  - Increase an AI’s attack power/capabilities/accuracy
  - Exploit built-in knowledge
    - Level geometry, shortcuts, the player’s location and inventory, etc.
- Practical tips
  - Build seemingly complex behaviours by combining simple reusable behaviours
  - Set a limit on AI attempts that fail
    - E.g., repeated collision with the same object, path to location cannot be found
  - Store history of an AI entity’s events
    - Useful for debugging, measuring and improving efficiency

Movement

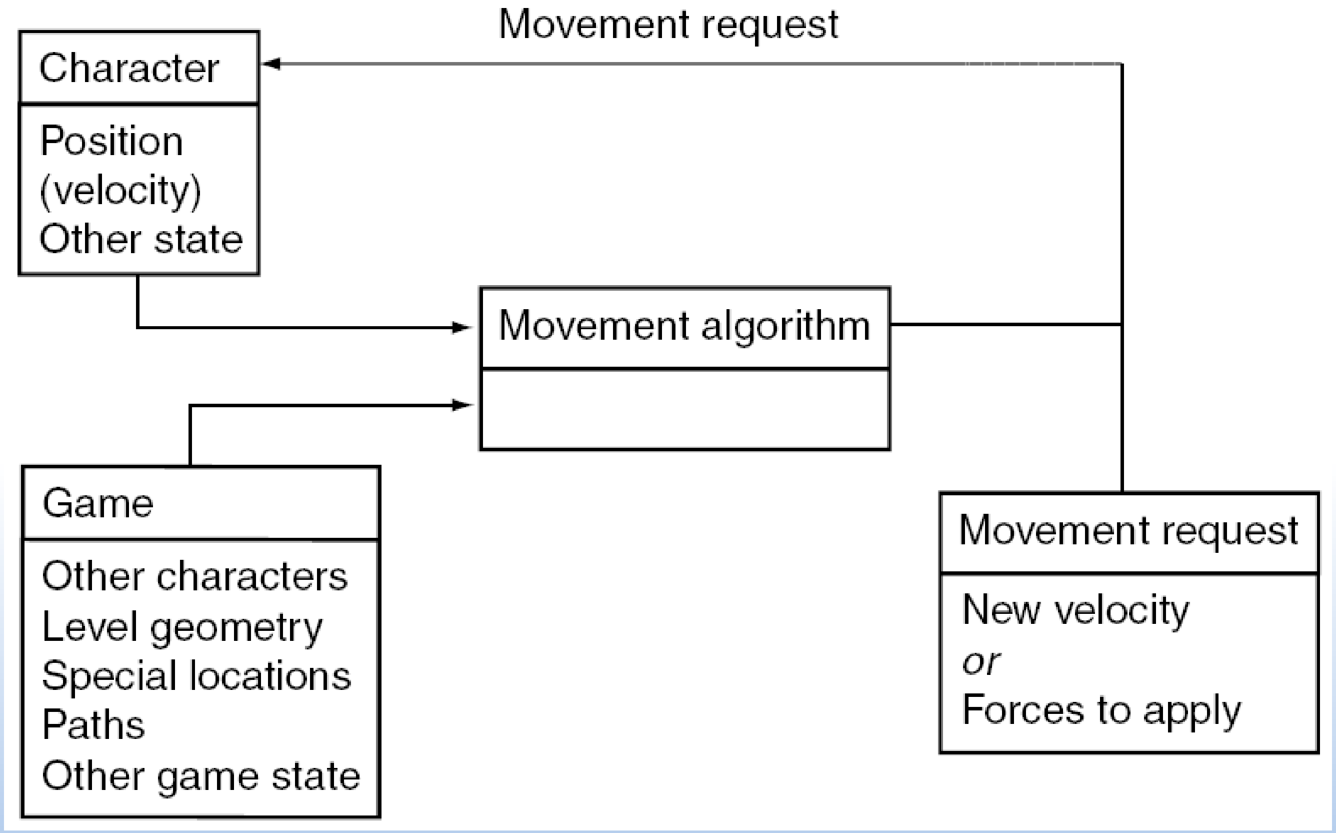
- Movement
  - ➢ One of the most fundamental requirements of AI
  - ➢ Some games rely solely on movement algorithms
    - May not require any advanced decision-making
- Autonomous agent
  - ➢ A system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future

Movement

- Movement
  - ➢ Can be broken down into
    - Action selection
      - The part of the agent’s behaviour responsible for choosing its goals and deciding what plan to follow
    - Steering
      - This layer is responsible for calculating the desired trajectories required to satisfy the goals and plans set by the action selection layer
    - Locomotion
      - Represents the more mechanical aspects of an agent’s movement

Movement Algorithms

- Movement algorithms
  - Structure of a basic movement algorithm

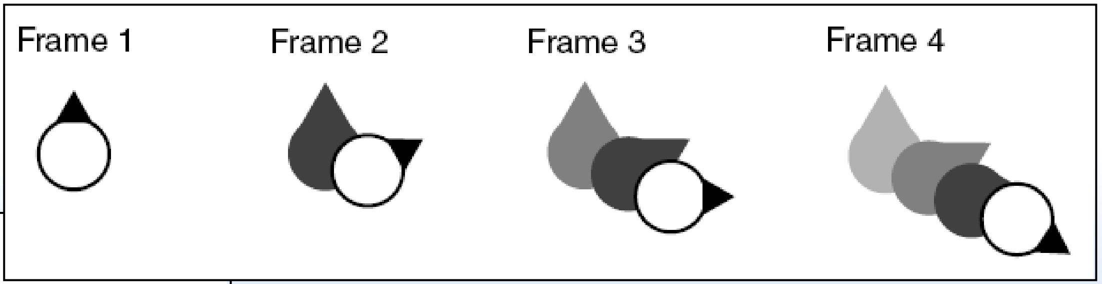


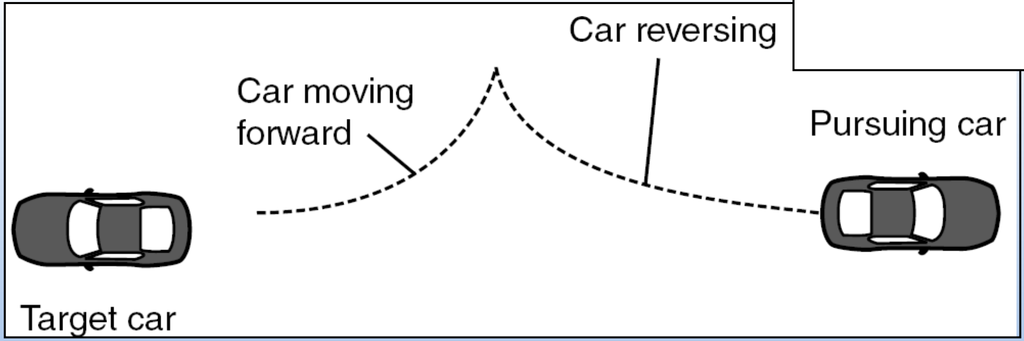
Movement Algorithms

- Agent movement data
  - ➢ Location, orientation, velocity, mass, etc.
  - ➢ Max/min values
    - Max speed, acceleration, turning rate, etc.
- 2D calculations
  - ➢ Many 3D games only implement AI calculations in 2D
  - ➢ Three/four degrees of freedom
    - *x*, *y*, (possibly *z*), and yaw

Movement Algorithms

- Facing direction
  - Need to connect this with the moving direction
  - Immediately changing an AI entity's orientation to face the direction of motion breaks the illusion of immersion
    - E.g., directly using the `Transform.LookAt()` method in Unity
    - The change needs to be over many frames



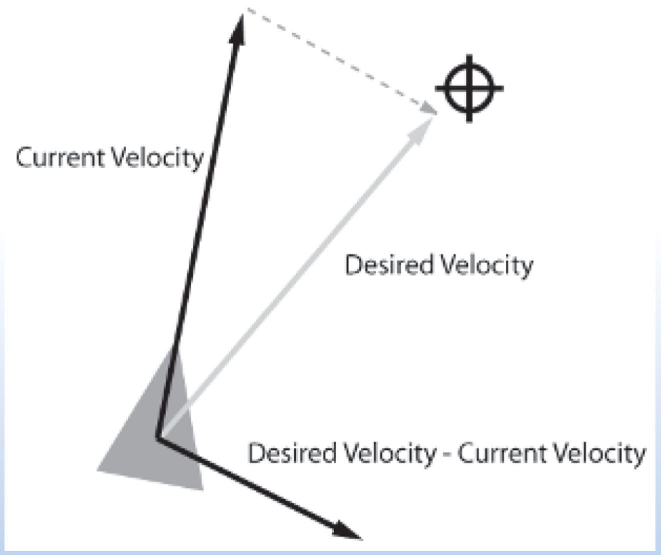


Movement Algorithms

- Unity
  - Interpolation
    - `Vector3.Lerp()`
      - Linear interpolation between two points
    - `Vector3.Slerp()`
      - Spherically interpolates between two vectors
    - `Quaternion.Slerp()`
      - Quaternion spherical interpolation
    - Accept two values `a` and `b` and an interpolant `t` clamped to the range [0, 1]
    - Returns a + (b - a) * t

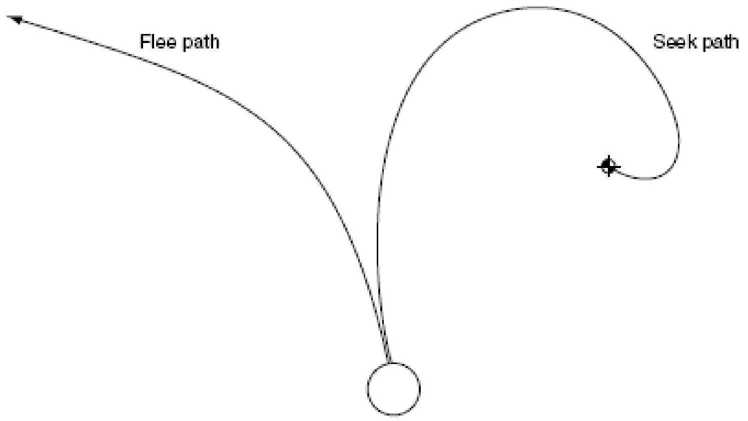
Steering Behaviours

- Steering behaviours
  - ➢ Commonly implemented in games
    - Simple components, quick to implement and compute
    - Simple to understand and implement but can produce seemly complex movement patterns
  - ➢ Seek
    - Tries to match the position of the agent with the target's position
    - Finds the direction to the target and heads toward it



Steering Behaviours

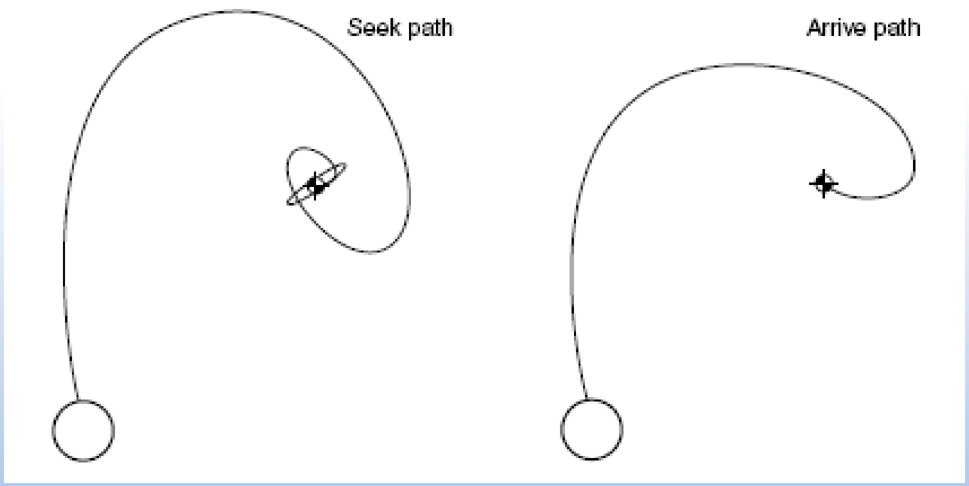
- Flee
  - Opposite of seek
  - Instead of steering agent toward target position
    - Reverse direction and make agent run away
  - Might only want to flee when the target is within a certain range



- Arrive
  - Seek is fine if the agent is chasing the target (constantly moving)
    - Doesn't stop gracefully, likely to overshoot an exact spot
    - May oscillate backward and forward on successive frames trying to get there

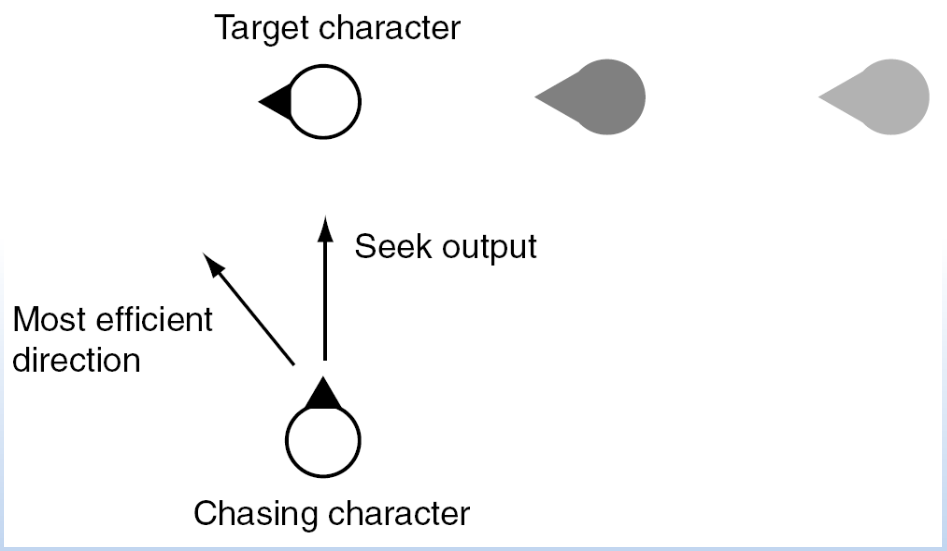
Steering Behaviours

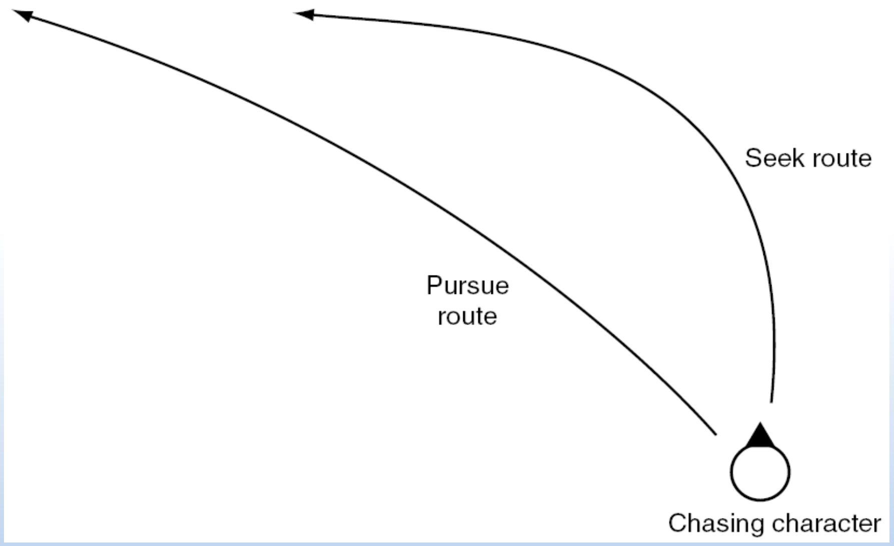
- To overcome this
  - Slow agent down (decelerate) as it reaches its target
    - » Might still overshoot by a shorter margin
  - Give it a large radius of satisfaction
  - Simple way
    - » Implement a fixed time/distance to target, when target is within a certain distance, reduce its speed



Steering Behaviours

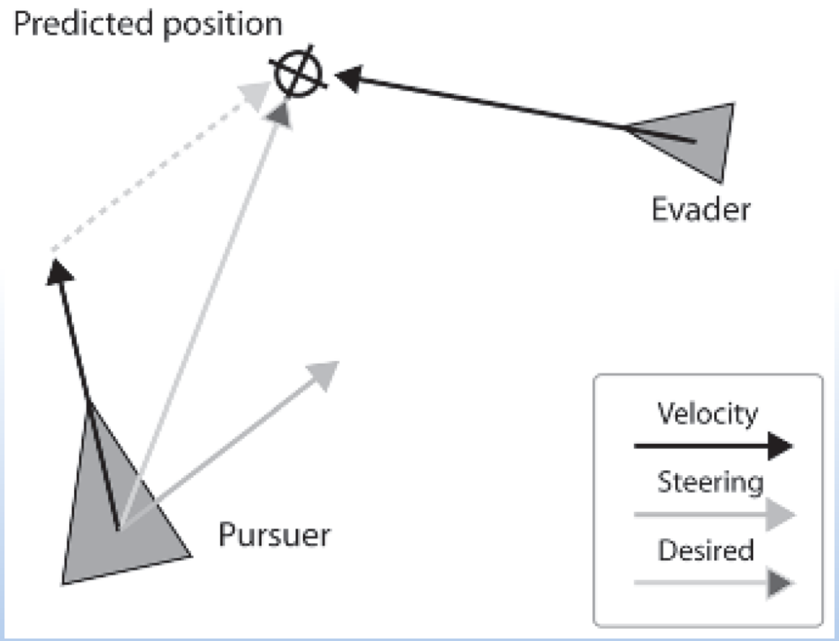
- Pursuit
  - Useful when agent required to intercept moving target
  - Seek only tracks current target position
  - Pursuit predicts where the target will be in future and aims there





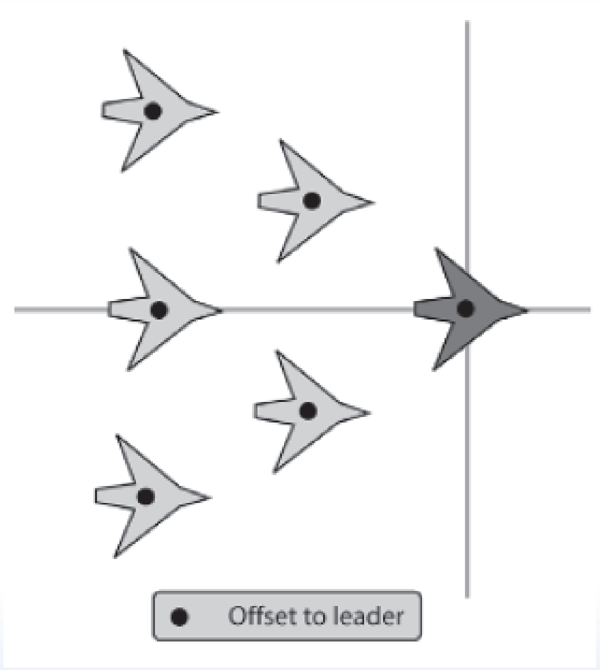
Steering Behaviours

- Can use a variety of complex prediction algorithms
- Simple way
  - Assume the target will continue moving with its current velocity
  - Use a prediction time interval to predict ahead
    - » Where the target will be given its current velocity



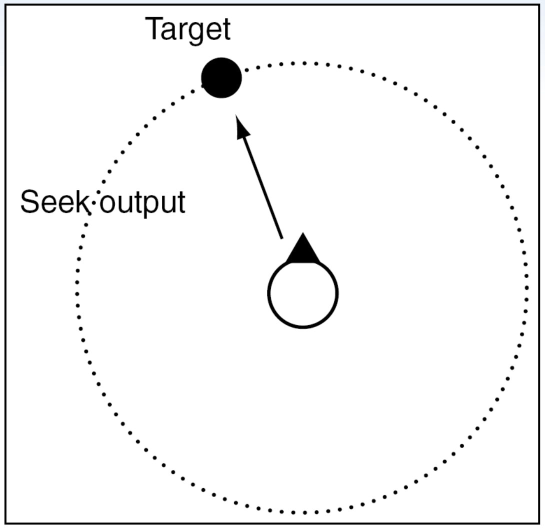
Steering Behaviours

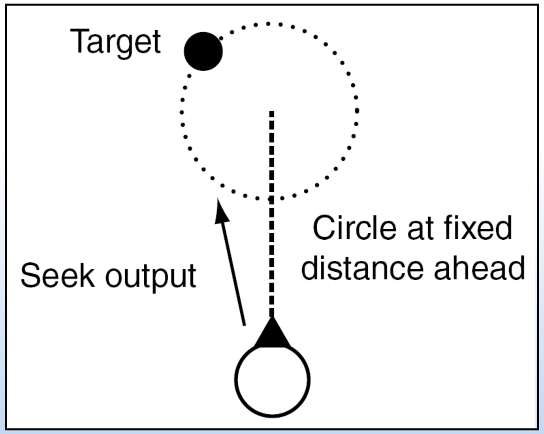
- Evade
  - Opposite behaviour to pursuit
  - This time evader flees from the estimated pursuer's future position
- Offset pursuit
  - Keep the agent positioned at a specified offset from the target
  - Useful for creating formations
  - Use in conjunction with the arrive behaviour instead of seek



Steering Behaviours

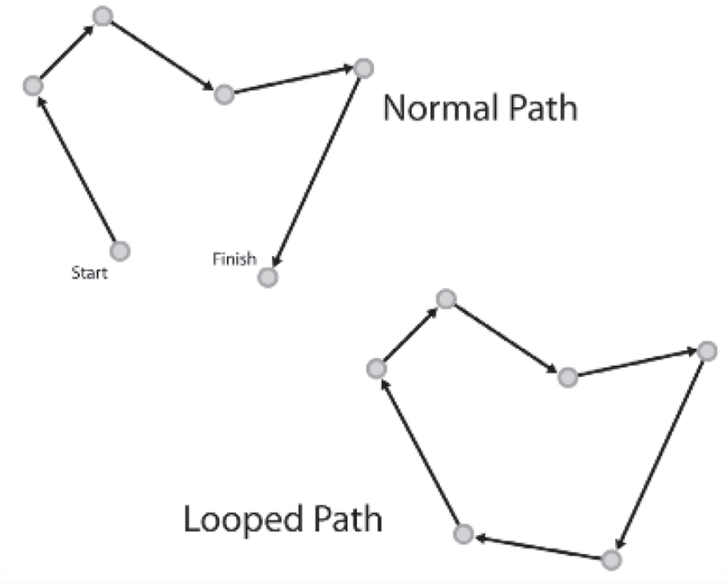
- Wander
  - Steering behaviour that gives the impression of random walk
  - Naïve approach
    - Random direction each time step
    - Produces jittery behaviour, no ability to achieve long persistent turns
  - Better approach
    - Move the circle out in front of the agent and shrink it down
    - Orientation changes a lot smoother

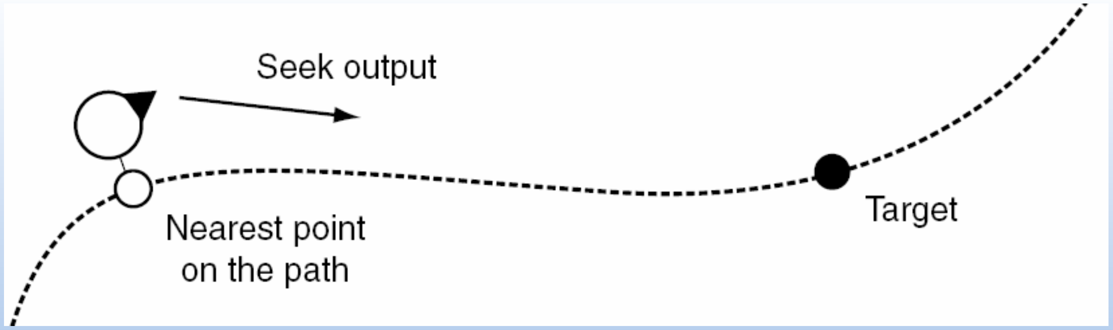




Steering Behaviours

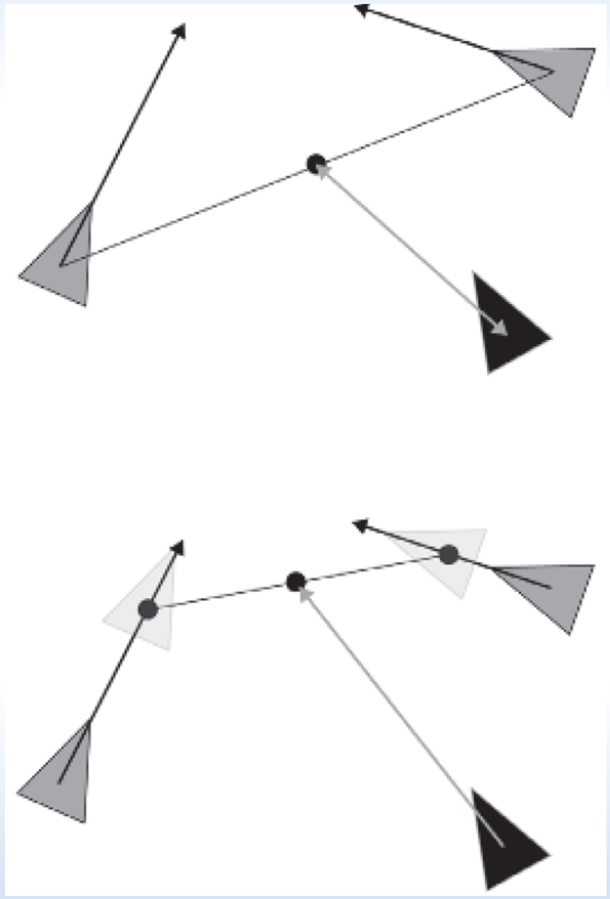
- Path following
  - Steering behaviour that takes a whole path as a target
  - Paths can be opened or closed
  - Seek target until within a certain radius, then get next target
  - For open paths, use arrive at last target





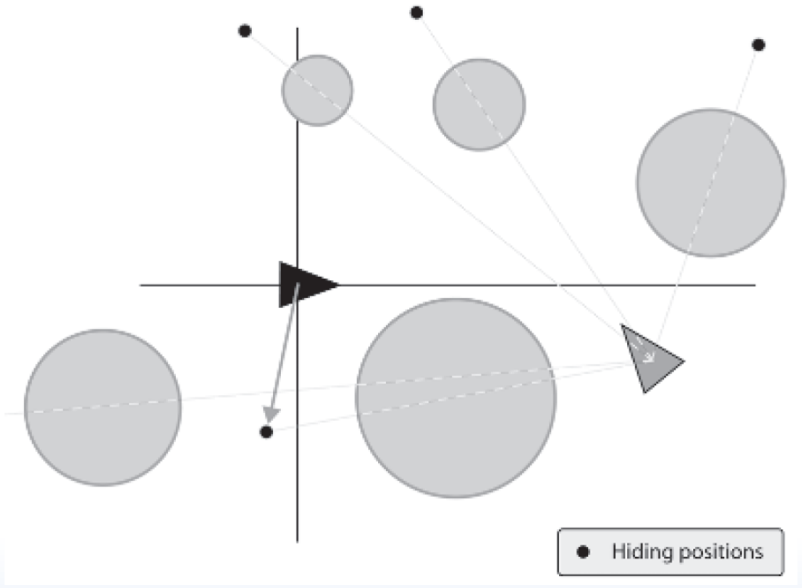
Steering Behaviours

- Interpose
  - Move the agent to the midpoint of an imaginary line between two targets
  - Like pursuit, must estimate the future position of the targets
- Hide
  - Attempts to position the agent so an obstacle is always between itself and the agent



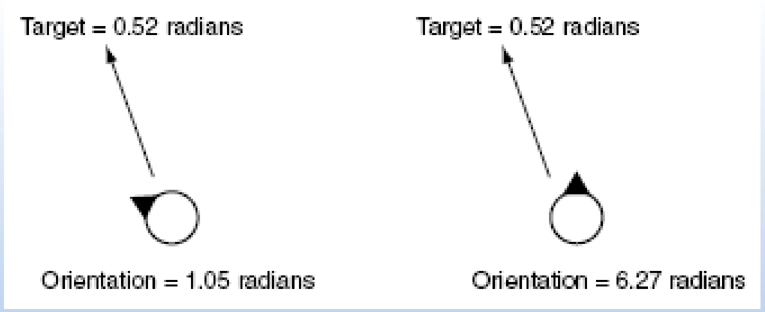
Steering Behaviours

- First, find a hiding spot for each obstacle
- Then, determine the distance to each spot
  - Use arrive to steer to the closest
  - If no appropriate hiding place is found
    - Use evade
- Might only want to hide if within the agent's field of view



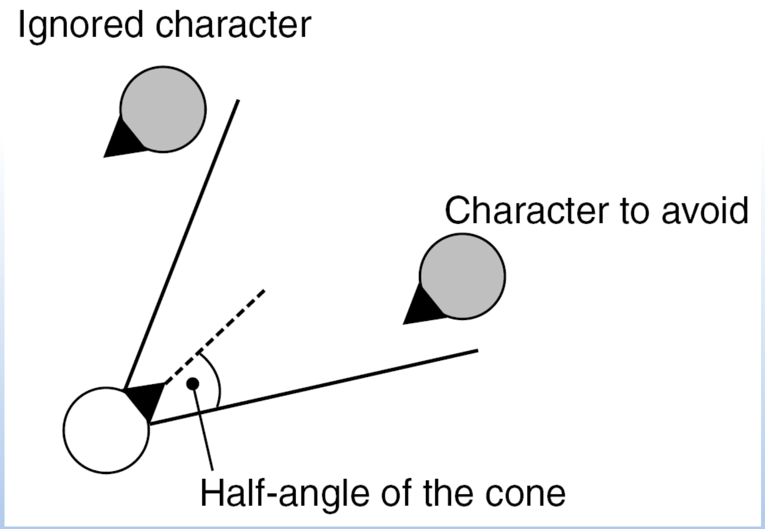
Steering Behaviours

- Separation (or repulsion steering)
  - Acts to keep the agent from getting too close to being crowded
  - Common in crowd simulation
  - Can define a threshold and use evade
- Align
  - Tries to match the orientation of the agent with that of the target
- Velocity matching
  - Tries to match a target’s velocity or velocity of the group
  - Useful in group behaviour



Collision Avoidance

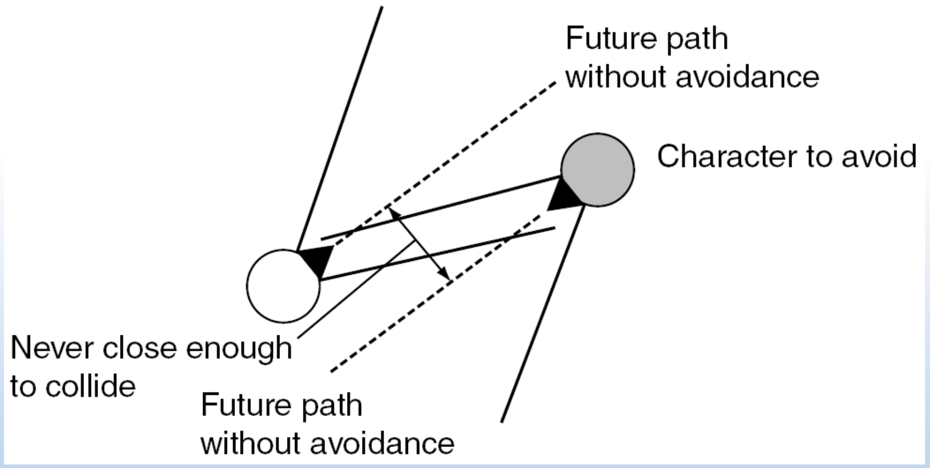
- Collision avoidance
  - ➢ Avoid other agents
    - Both moving and stationary agents
  - ➢ Simple approach
    - Use a variation of evade or separation behaviour, that only engages if target within a cone



Collision Avoidance

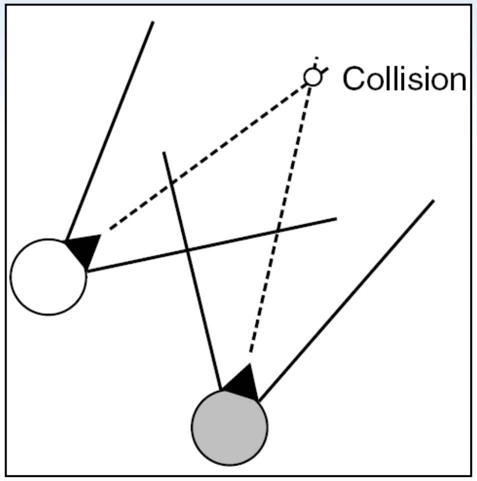
➢ Problem

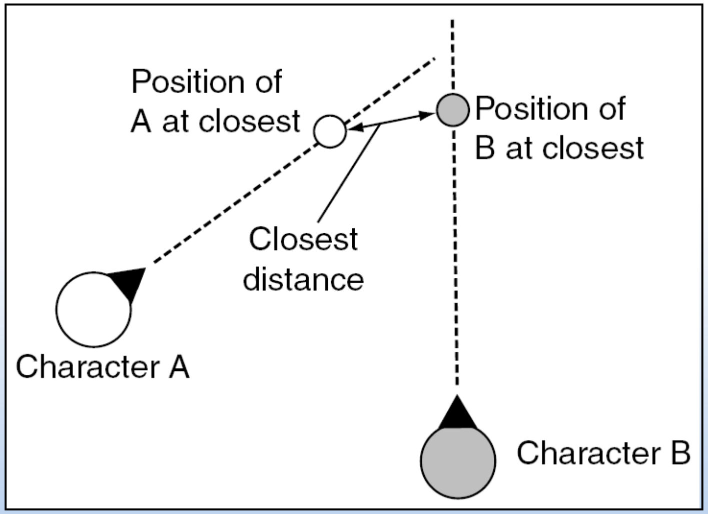
- If several targets within the cone, need to avoid all
  - Often sufficient to find average position and speed of all targets in cone, and evade that
  - Alternatively, find closest target and ignore the rest
- Unfortunately, in this approach agent will take action even if it might not actually collide



Collision Avoidance

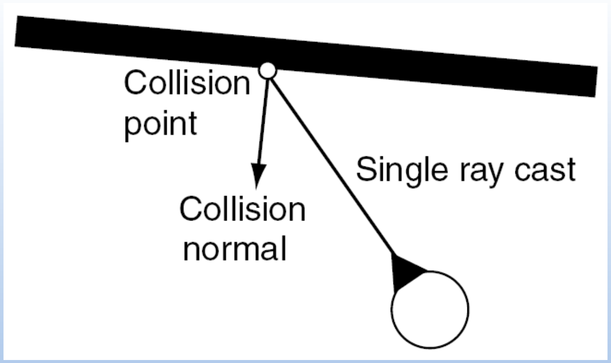
- Another problem
  - Situation where an agent might collide with targets outside the cone
  - Better solution
    - Use collision prediction
    - Work out whether or not agents will collide if they keep their current velocity
    - Evasion based on predicted future positions rather than current positions





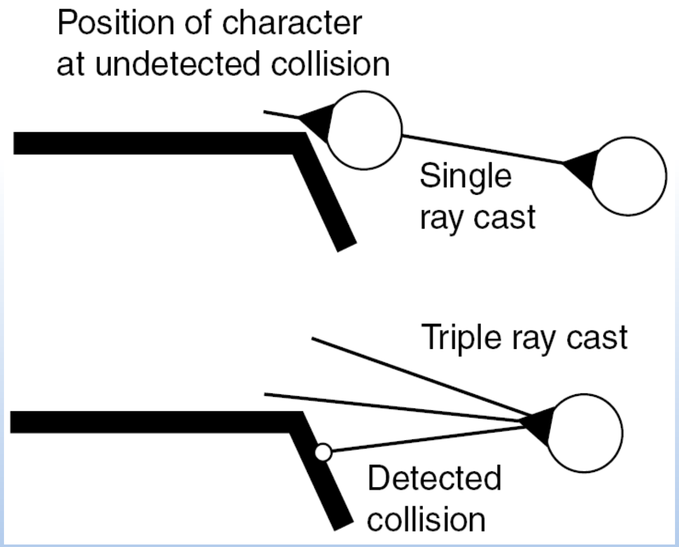
Collision Avoidance

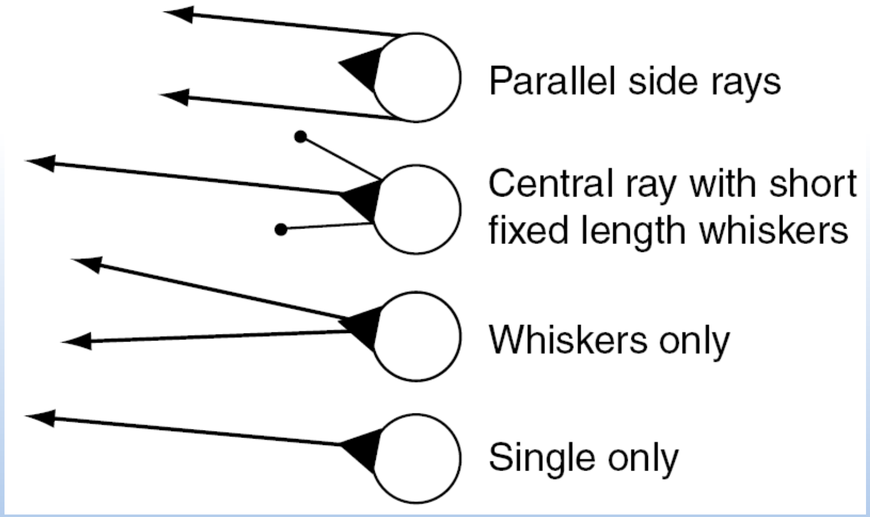
- Obstacle and wall avoidance
  - ➢ The previous approach
    - Only works for agents or objects that can comfortably fit within a bounding sphere
  - ➢ To avoid large obstacles
    - Agent casts one or more rays out in the direction of motion
    - Short rays can be used
    - If ray collides with obstacle, then a target is created that has to be avoided



Movement Algorithms

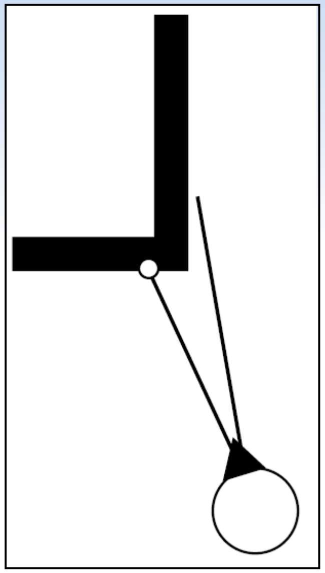
- Problems
  - Using a single ray runs into some problems
    - Not enough precision, may collide with corners
    - Can use whiskers to avoid this
  - Whiskers can sometimes make it impossible to move down tight passages

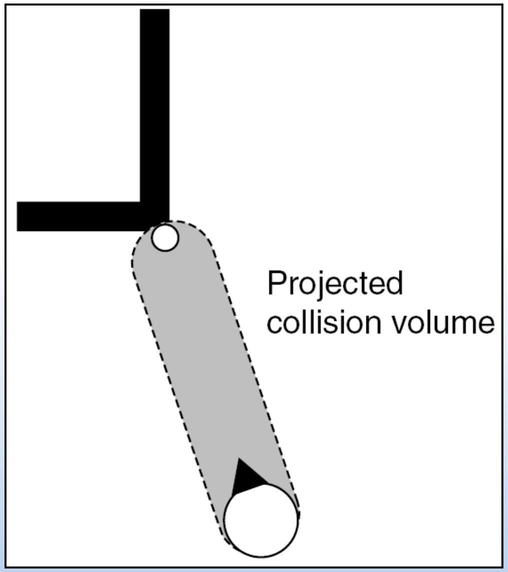




Collision Avoidance

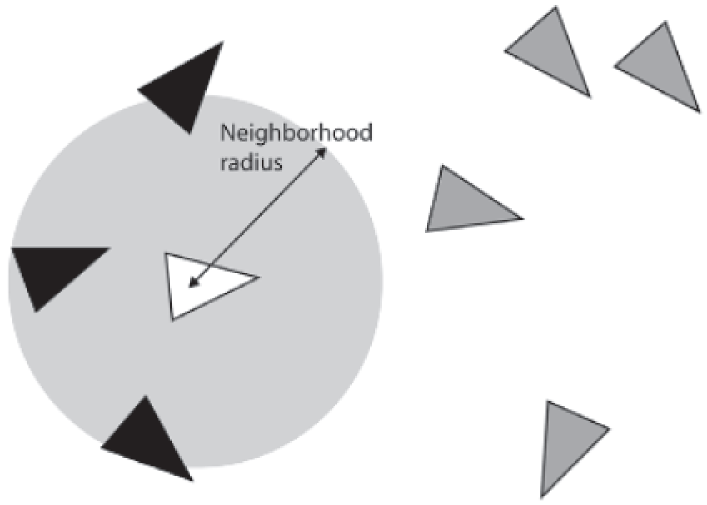
- Multiple rays can suffer from the corner trap
  - Can use adaptive fan angles
    - » Angle widens when collision detected, otherwise narrowed
  - Or ignore some rays
- The most practical solution
  - Use projected collision volume, but complex and can be very slow for steering behaviours





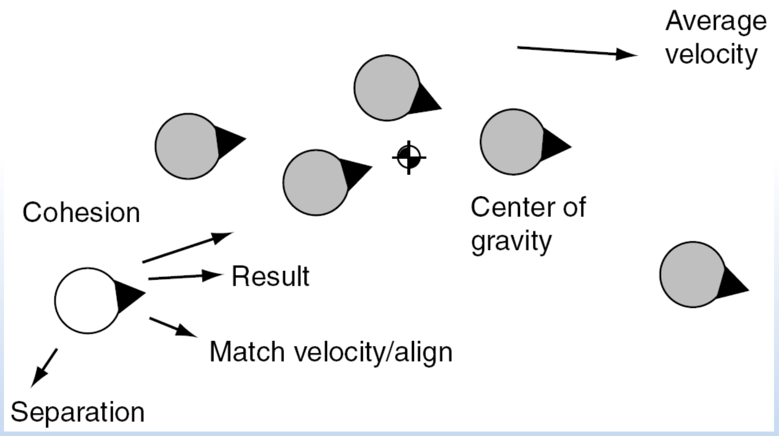
Combining Steering Behaviours

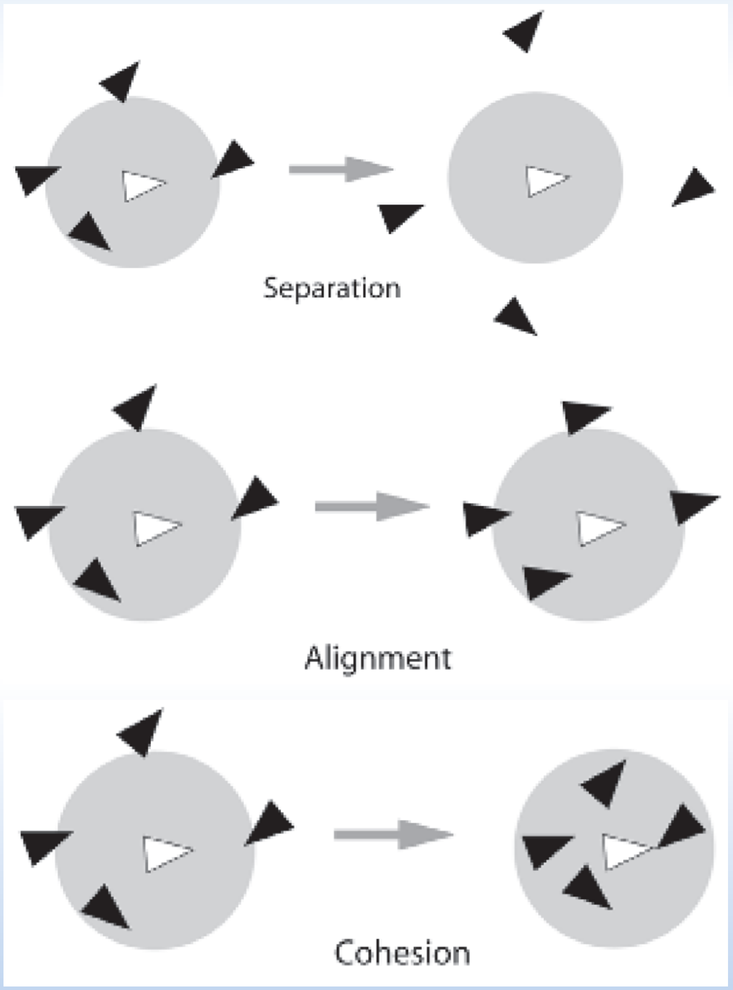
- Group behaviour
  - ➢ Steering behaviour that
    - Takes into consideration some, or all, other agents in the game world
    - Agent will consider all other agents within a circular area
  - ➢ Flocking and swarming
    - The flocking algorithm relies on blending three steering behaviours
      - Separation
      - Alignment (and velocity matching)
      - Cohesion



Combining Steering Behaviours

- Separation
- Alignment (and velocity matching)
- Cohesion
  - Move toward centre of mass of flock



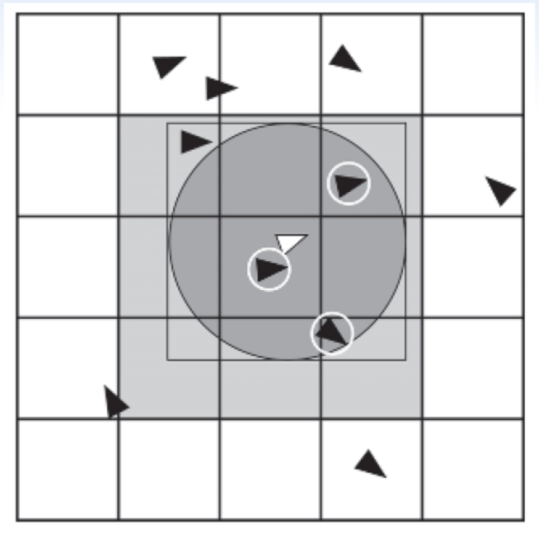


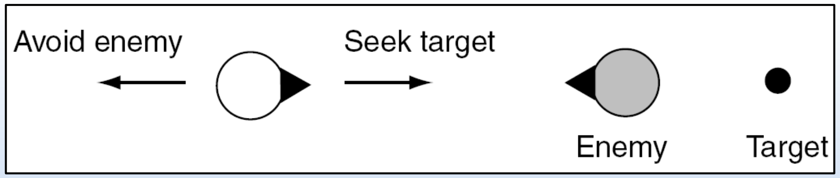
Combining Steering Behaviours

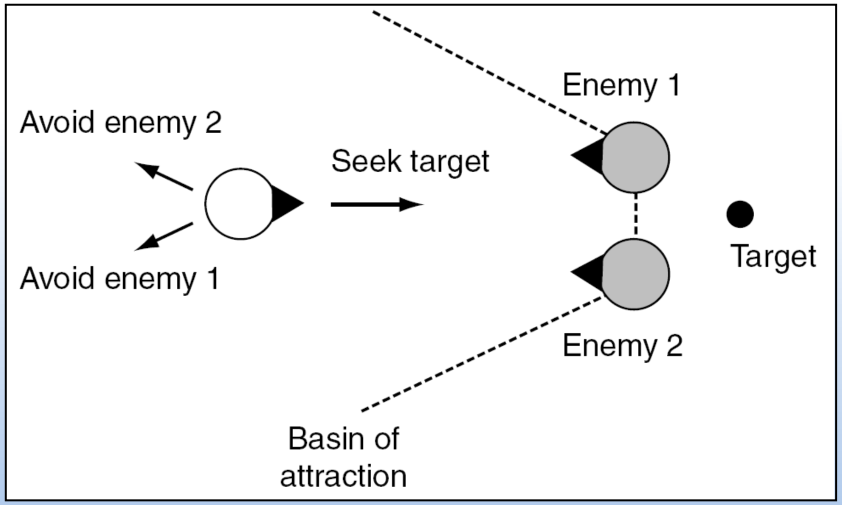
- Blended steering behaviour
  - Can achieve more complex movement
  - Combine steering behaviours using a set of weights and/or priorities
    - Weighted truncated sum
      - Combine each steering behaviour by a weighted sum, then truncate by maximum allowable velocity or acceleration
    - Priorities
      - Some steering behaviours more important than others
    - Weights or priorities may change over time, or may even be switched off

Combining Steering Behaviours

- Spatial partitioning
  - BSP trees, quadtrees, octrees, grids
  - Improves efficiency of finding neighbours
- Problems
  - In outdoor environments
    - May get stuck when steering behaviours want to do conflicting things

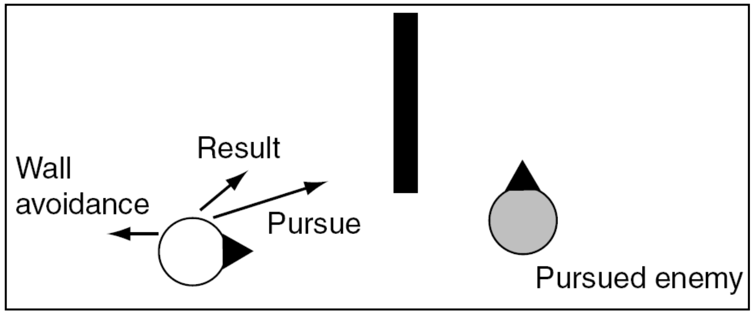


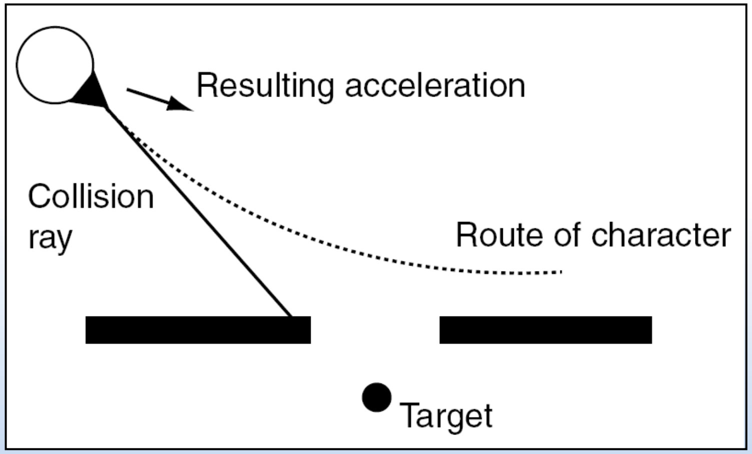


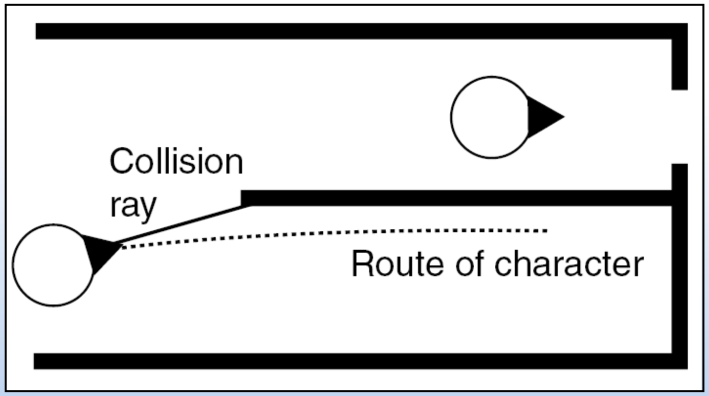


Combining Steering Behaviours

- Other problems
  - Constrained environments
    - Movement in indoor environments greatly constrains an agent's movements
  - Nearsightedness
    - Steering behaviors act locally based on immediate surroundings

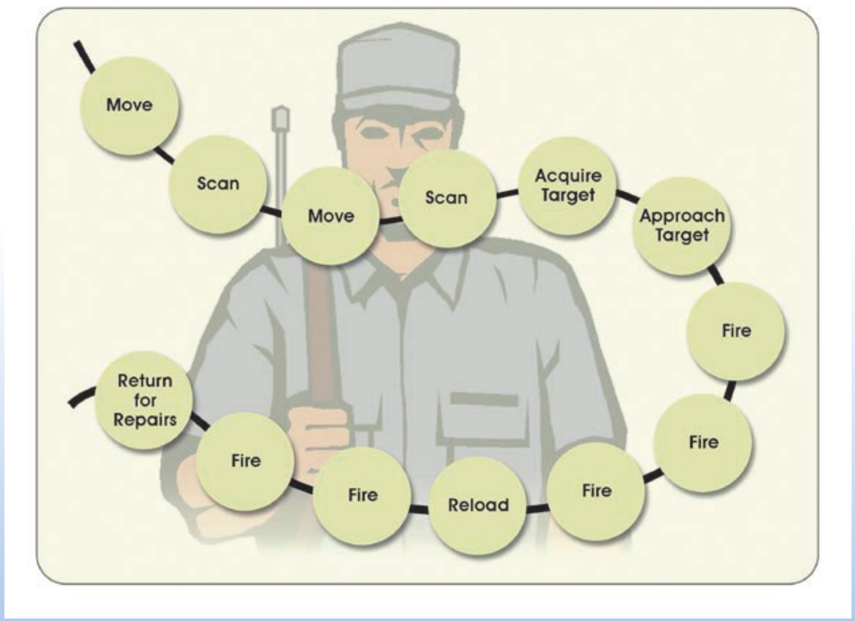


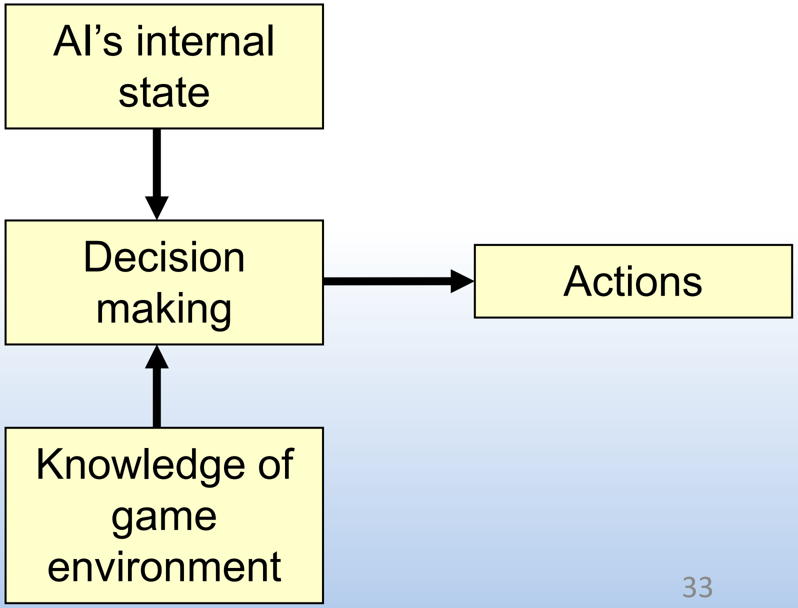




Controlling Behaviour

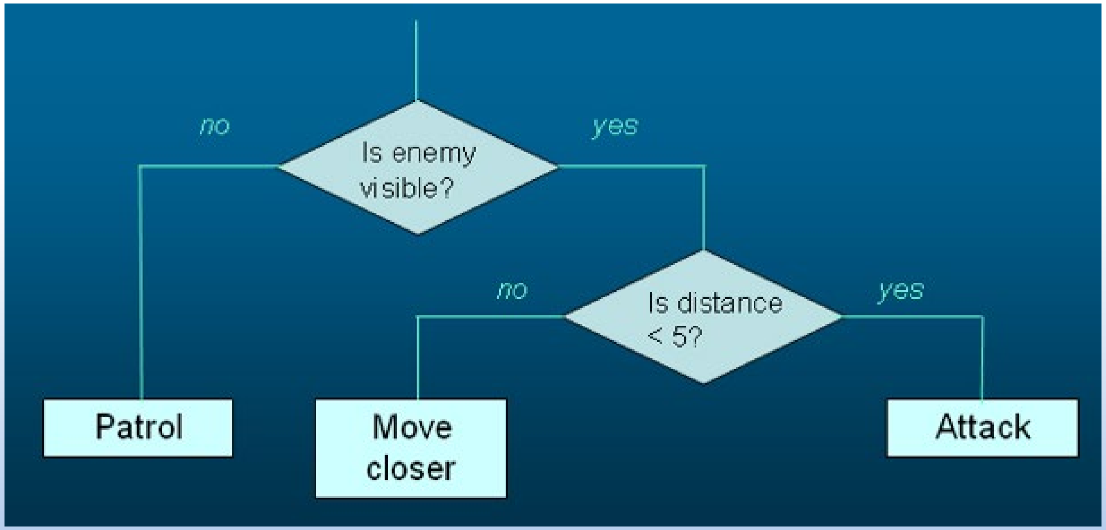
- Behaviour
  - ➢ A context specific sequence of actions
  - ➢ Dynamic sequences
    - Actions in the sequence should vary depending on circumstances (internal state and relevant game environment)





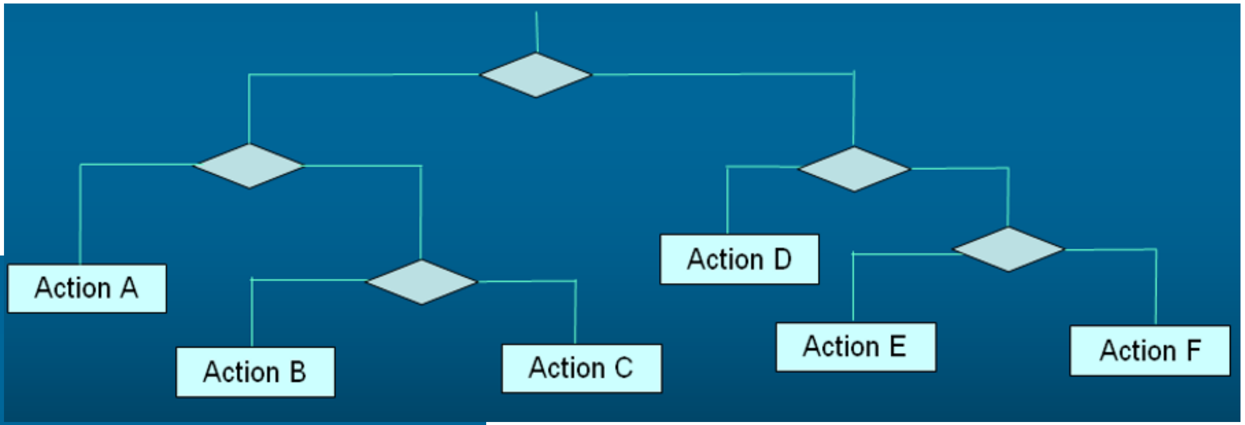
Controlling Behaviour

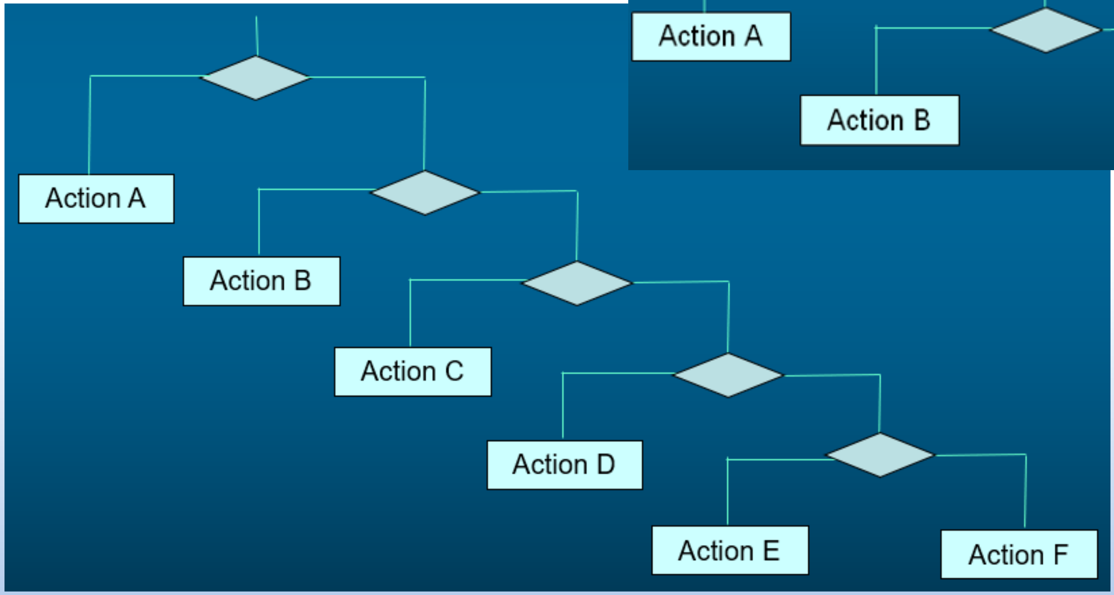
- Decision trees
  - ➢ Connected decision points that use an if-else structure
  - ➢ An algorithm goes along the tree to make choices at each decision node
    - When the algorithm gets to an action, the action is performed immediately



Controlling Behaviour

- Tree balancing
  - The longer the branch the longer it takes to process



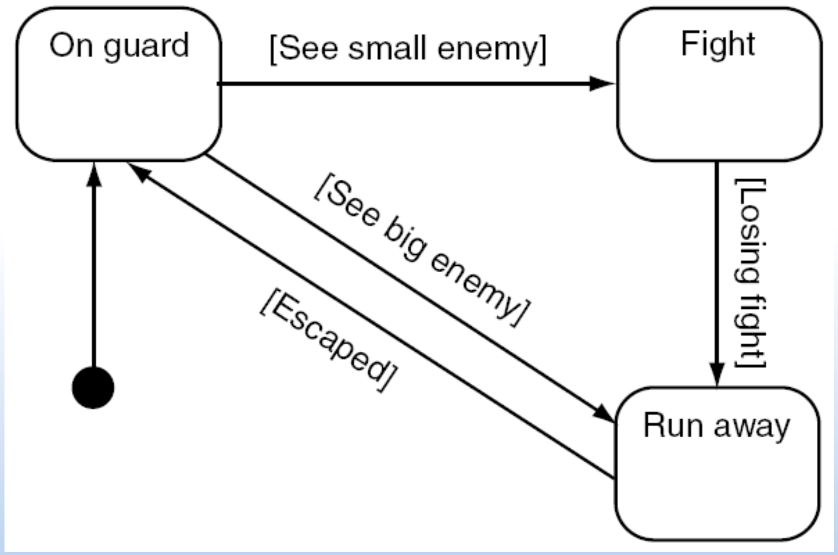


Controlling Behaviour

- Finite state machines
  - ➢ Most common behavioural modelling algorithm
    - Conceptually simple but can create flexible AI with little overhead
  - ➢ Consists of
    - A number of states
      - – Normally, actions or behaviours associated with each state
      - – Only one state is “active”
      - – Connected together by transitions
    - Transitions between those states
      - – Each transition leads from one state to another
      - – Each has a set of associated conditions, if condition met, new state becomes active

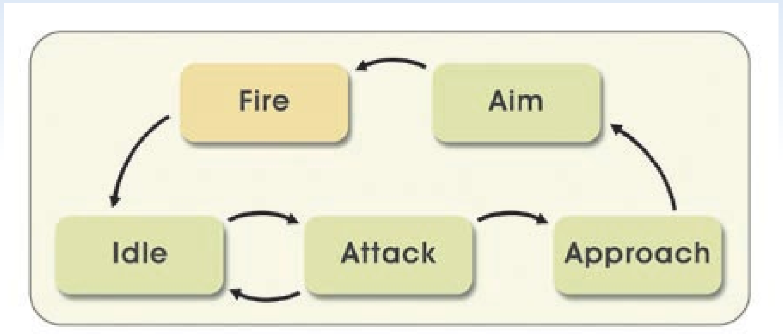
State Machines

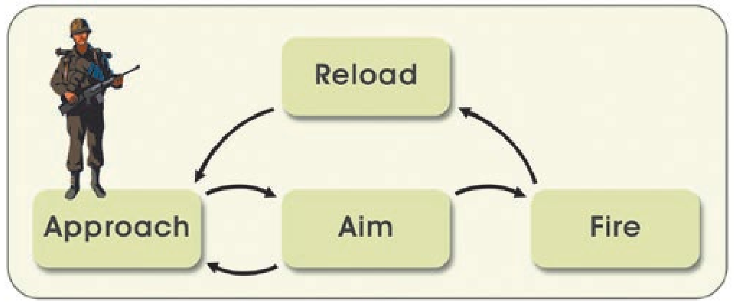
- One state active at any given time
  - Active state performs its actions
  - All other states inert
  - Continues in this state until transition occurs

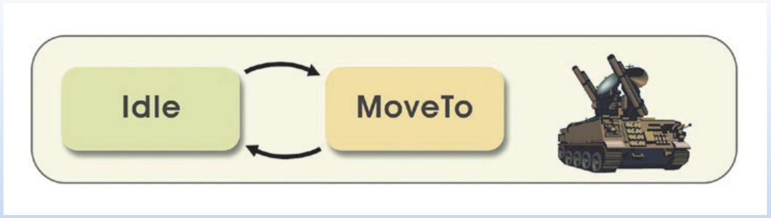


State Machines

- ➢ State transitions
  - Can be one way or both ways
  - Can be generated internally or externally
  - Internal
    - – Active state generates a transition to another state
  - External
    - – Transition in response to system or player commands







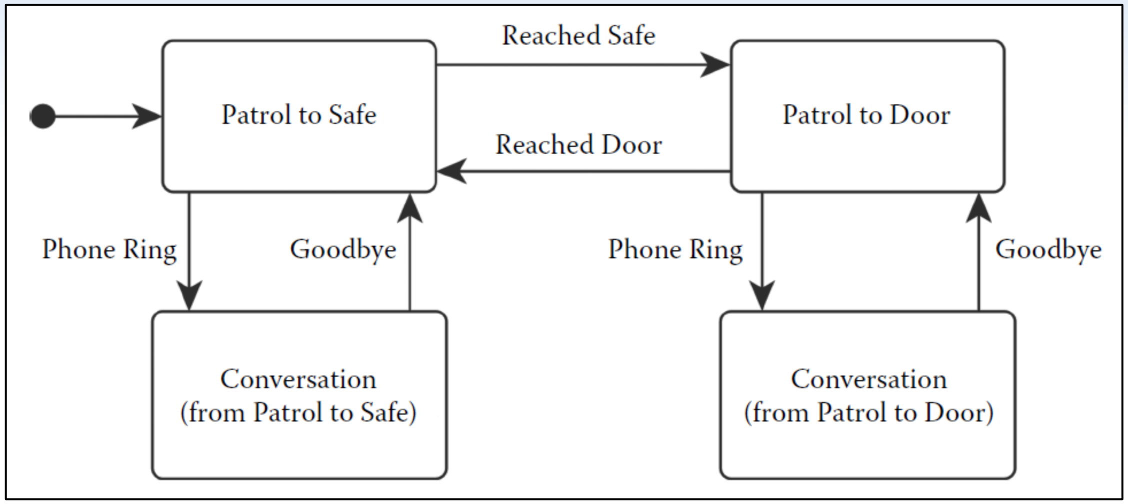
State Machines

- Hierarchical state machines
  - ➢ Complex games
    - Have over 100 states
    - Complex behaviours might be implemented across a number of states
    - Difficult to manage and error-prone
  - ➢ Hierarchical state machines
    - Useful when dealing with a large number of states
    - Define top level states
      - – Then move into sub-states to elaborate details
    - Easier to understand
      - – Less states at top level, more efficient
      - – Potentially result in faster game performance

State Machines

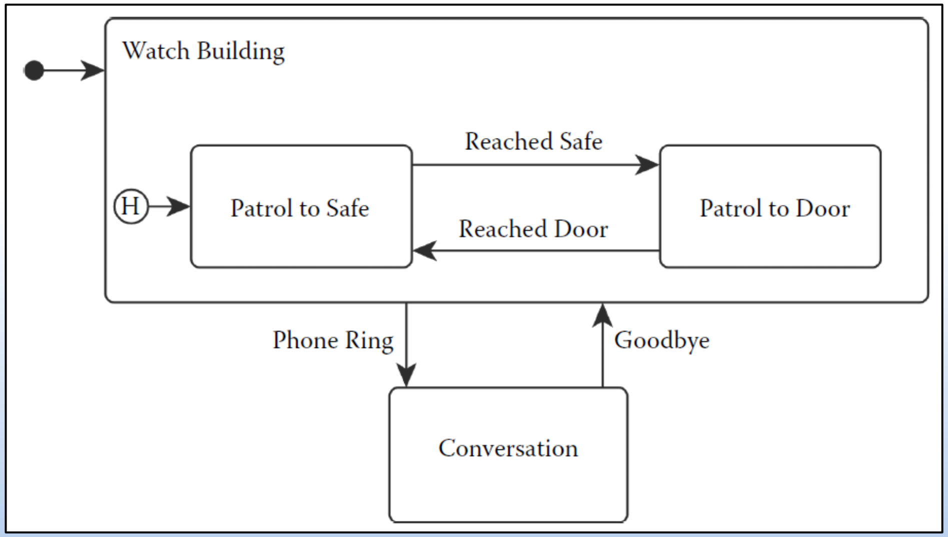
Finite state machine

- Duplicate *Conversation* states
- Difficult to reuse behaviour



Hierarchical state machine

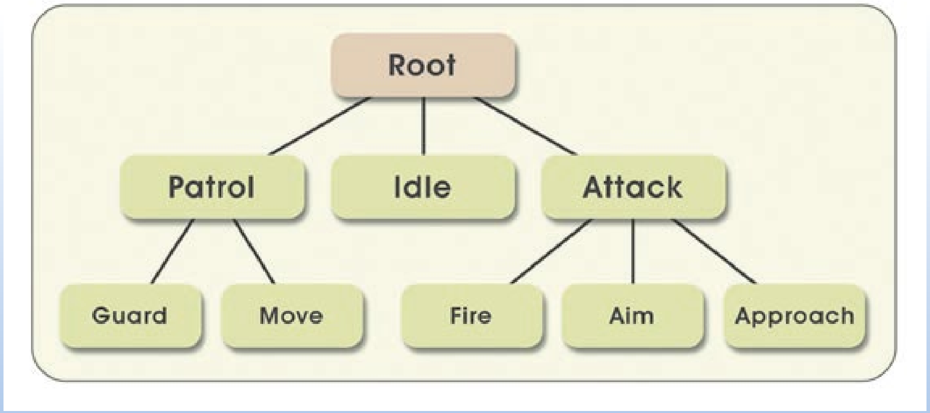
- Avoids duplicate *Conversation* states
- Nested state machine

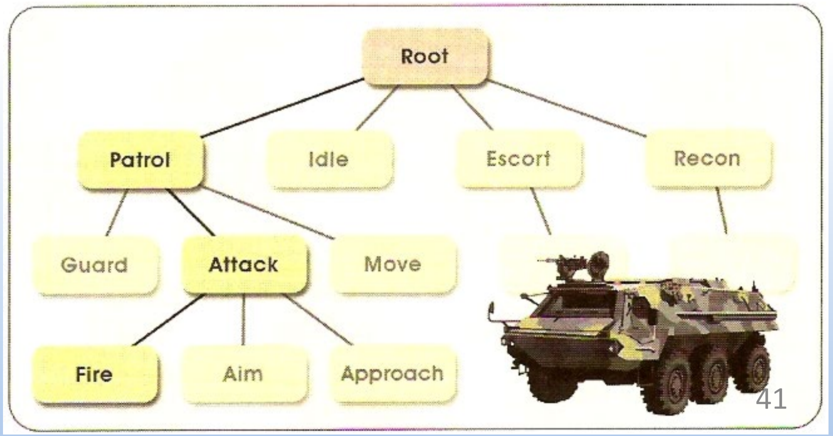


M. Dawe et al., Behavior Selection Algorithms: An Overview, Game AI Pro

State Machines

- Hierarchical state machine
  - ➢ Root machine has a list of states
  - ➢ Each state may have one or more sub-states
  - ➢ At any given time
    - One state is active
    - If has sub-states, then one sub-state active
  - ➢ Reduces number of states at top level





State Machines

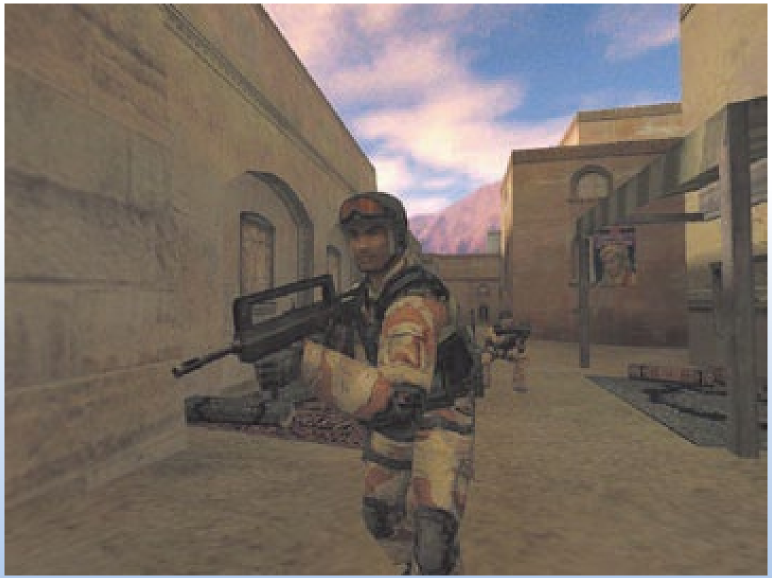
- Advantage of state machines
  - ➢ Only one state active at a time
  - ➢ Limited to a single action
    - Prevents going to two locations at once or firing a weapon at two different targets simultaneously
- Problem
  - ➢ How to handle
    - Interrupted goals
    - More than one goal at a time

State Machine Models

- Interrupted goals
  - ➢ State suspend and continue
    - Allow state to be suspended
    - Execute another state until finish
    - Return to original (interrupted) state
  - ➢ State stack
    - Has multi-level suspend and continue capabilities
    - Provides push-state and pop-state mechanisms

State Machine Models

- More than one goal
  - Multiple state machines
    - Relatively simple way to support more than one goal
    - Need to make sure states in the multiple state machines do not conflict
    - Might limit the state machines into separate parts



References

- Among others, material sourced from
  - https://docs.unity3d.com
  - Jason Gregory, Game Engine Architecture, A.K. Peters
  - Ian Millington, Artificial Intelligence for Games, Morgan Kaufmann
  - Mat Buckland, Programming Game AI by Example, Wordware Publishing
  - John Alquist and Jeannie Novak, Game Development Essentials: Game Artificial Intelligence, Delmar Publishing
  - Screenshots are from various games, mainly sourced from www.ign.com and www.mobygames.com