COMP 476

Advanced Game Development

Session 1
Introduction to Game Architecture

Based on McShaffry Chapter 2; AI for G, Millington Chapters 1, 2.1-2.2, 2.4, 3.1-3.3 (Adapted from Dr. Feven's Slides)

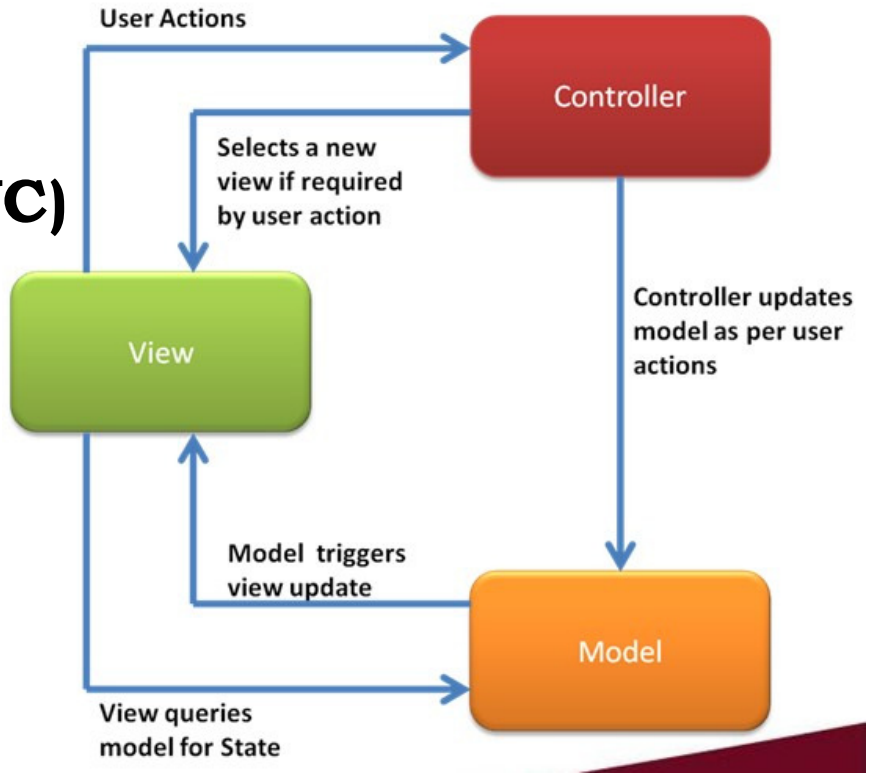
Lecture Overview

- High Level View of Game Architecture
- Game AI Fundamentals
- Movement AI

Game Architecture

Common Architectures:

- Microsoft Foundation Classes (MFC)
- Model-View-Controller(MVC)
- All are different ways of implementing the idea of Separation of Concerns
- For games architecture, a layer is added for O.S./hardware



Game Architecture

High level Architecture:

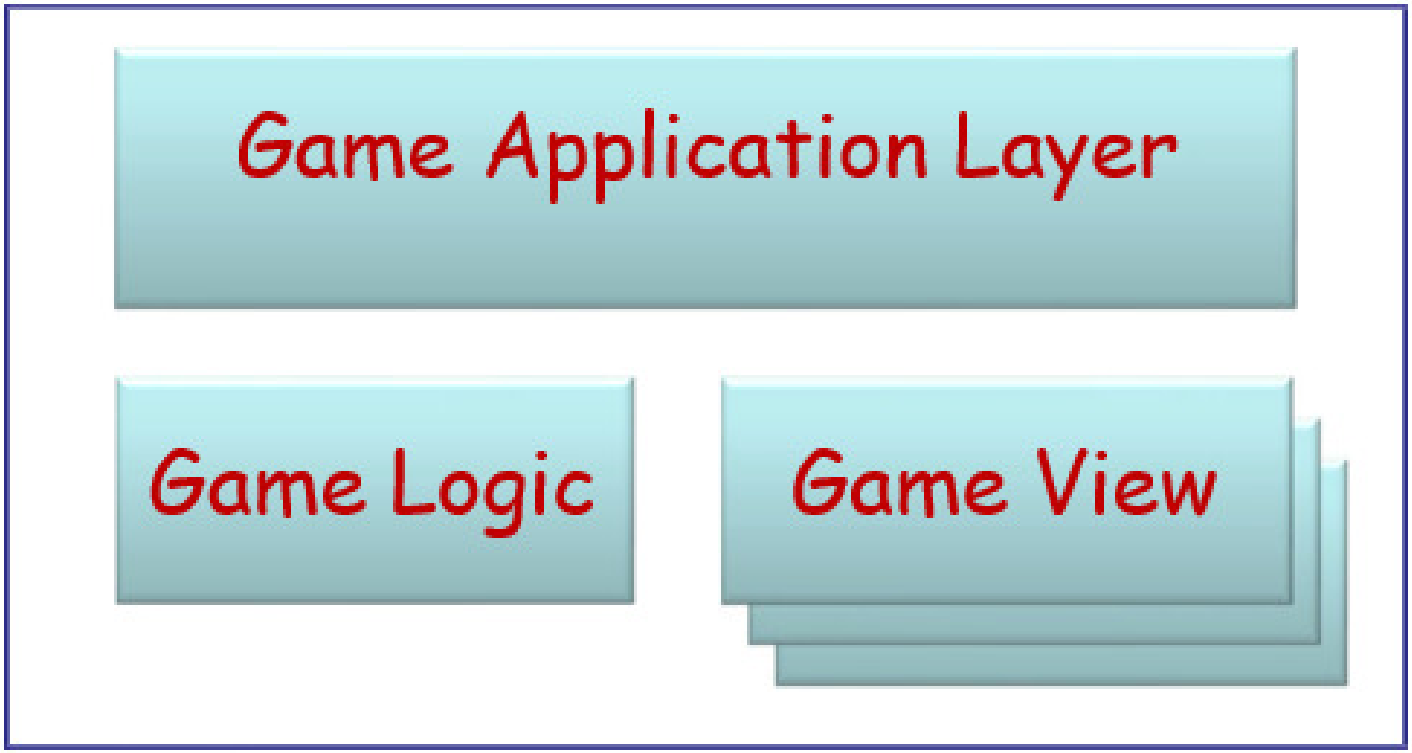
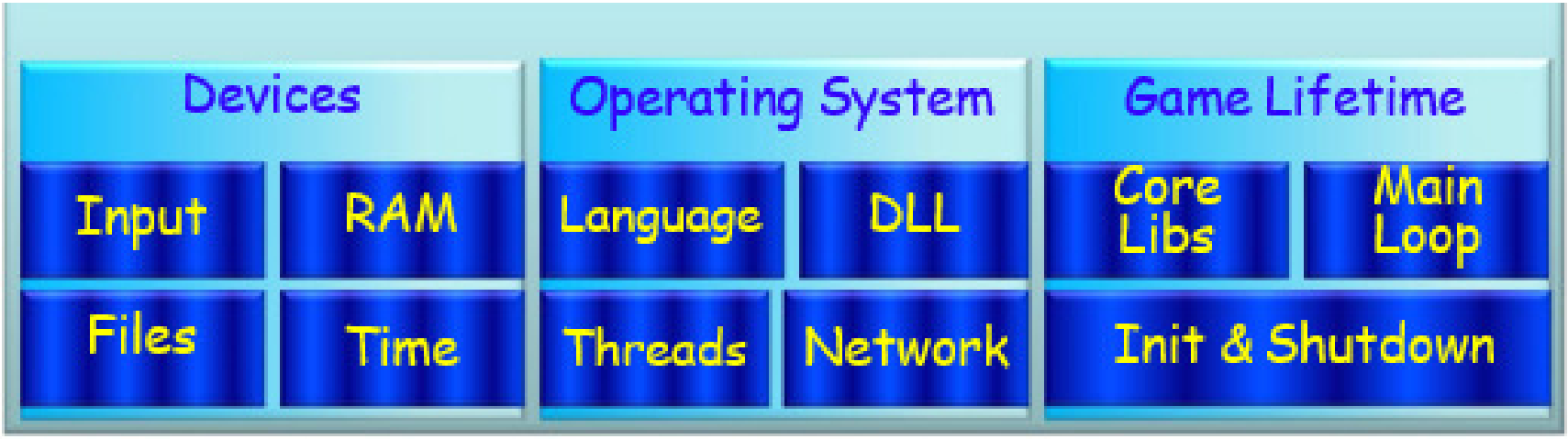


Figure from *Game Coding Complete,4th Ed.*, by Mike McShaffry

Game Architecture

Game Application Layer:



- Incorporates the machine/operating system
- Translates inputs into events handled by game logic and game views.

Game Architecture

Game Logic Layer:

- Game Logic: Game world's state and how it changes over time. "The heart and soul of your game"
- Game State: container for game objects.
- Integrated subsystems for:
  - Managing world state
  - Communicating state changes
  - Accepting input from other systems
  - Enforcing game rules such as Physics

Game Logic

- Data (game state)
  - Car
    - Weight distribution
    - Engine performance
    - Tire performance
    - Etc.
  - Track
    - Shape
    - Surface Properties

Image from: www.racer.nl

Game Logic

- Physics System
  - What happens while accelerating
  - How does the car respond to a track
- Collision Handling

Image from: www.racer.nl

Game Logic

- Input
  - Steering
  - Acceleration
  - Braking
  - Emergency Brake
- Physics System
  - Position, orientation of car
  - Position, orientation of wheels
  - Damage statistics
  - Collision events

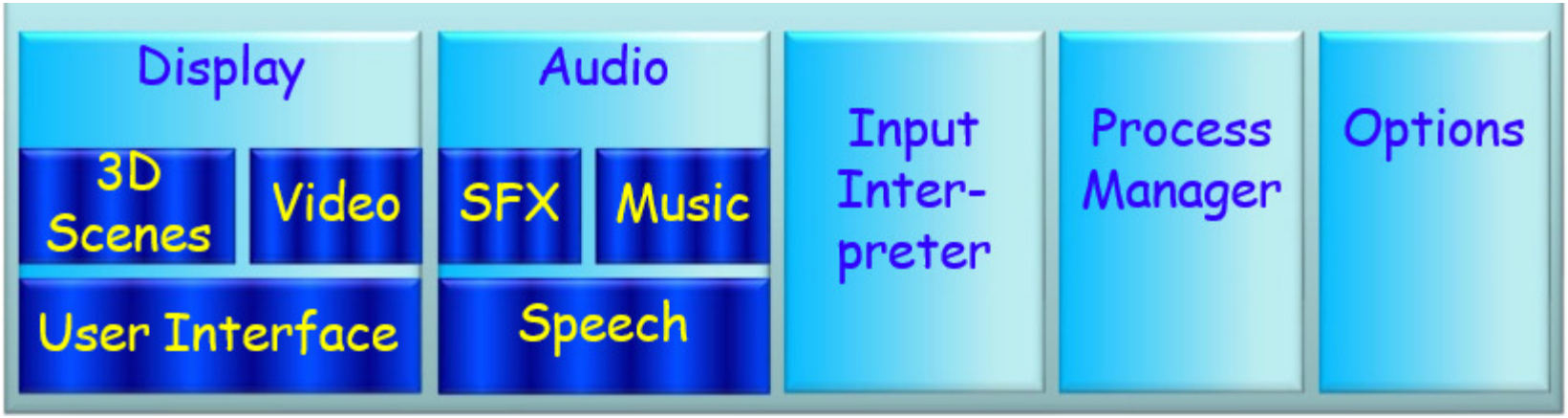
Image from: www.racer.nl

Game View

- Presents the game state
- Translates input into game commands for the game logic system
- Your game can have as many views as you want (or your computer can handle)
  - Human player view
  - AI agent view
  - Remote player view

Game View

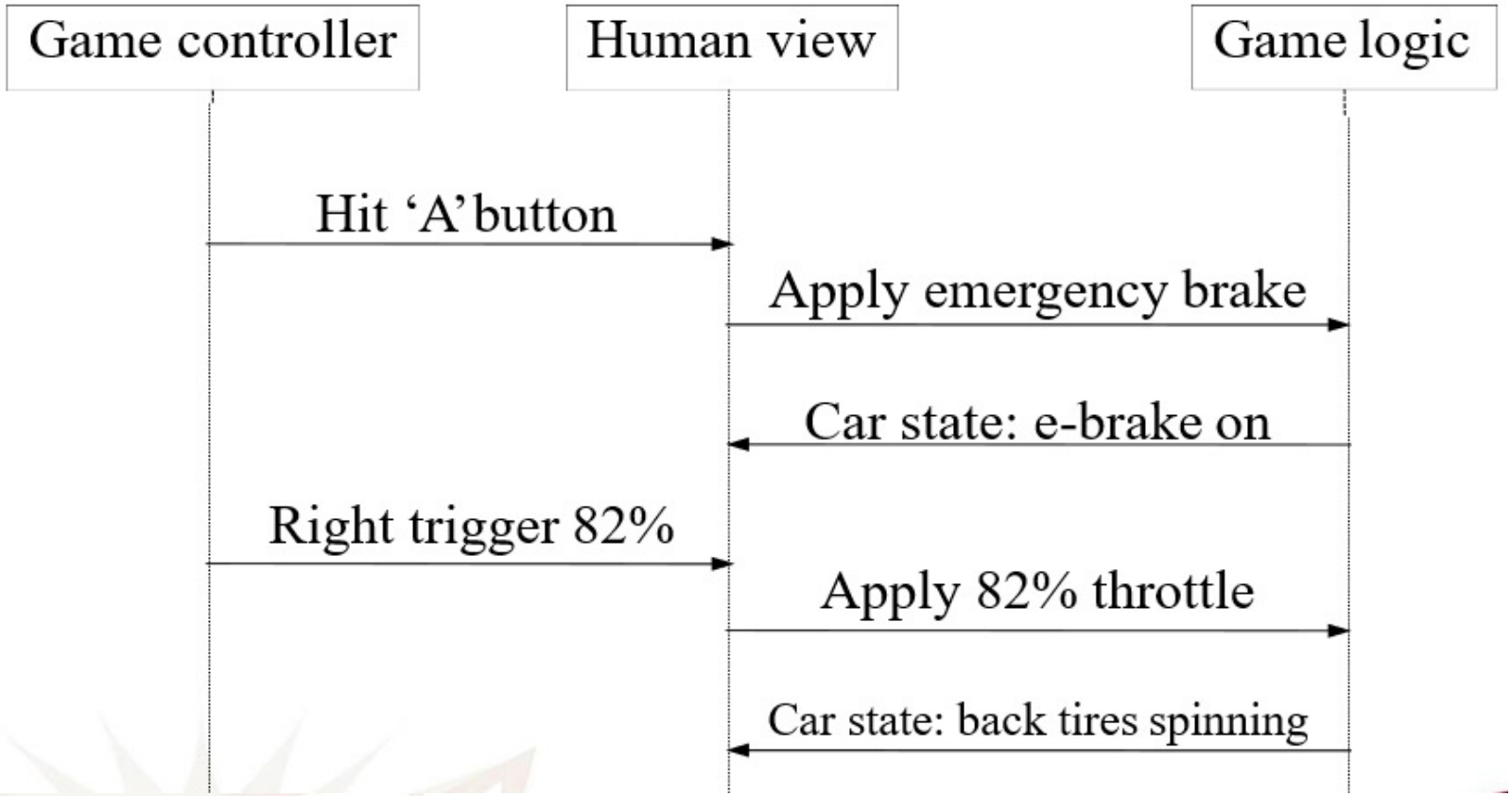
Human Player



AI Agent

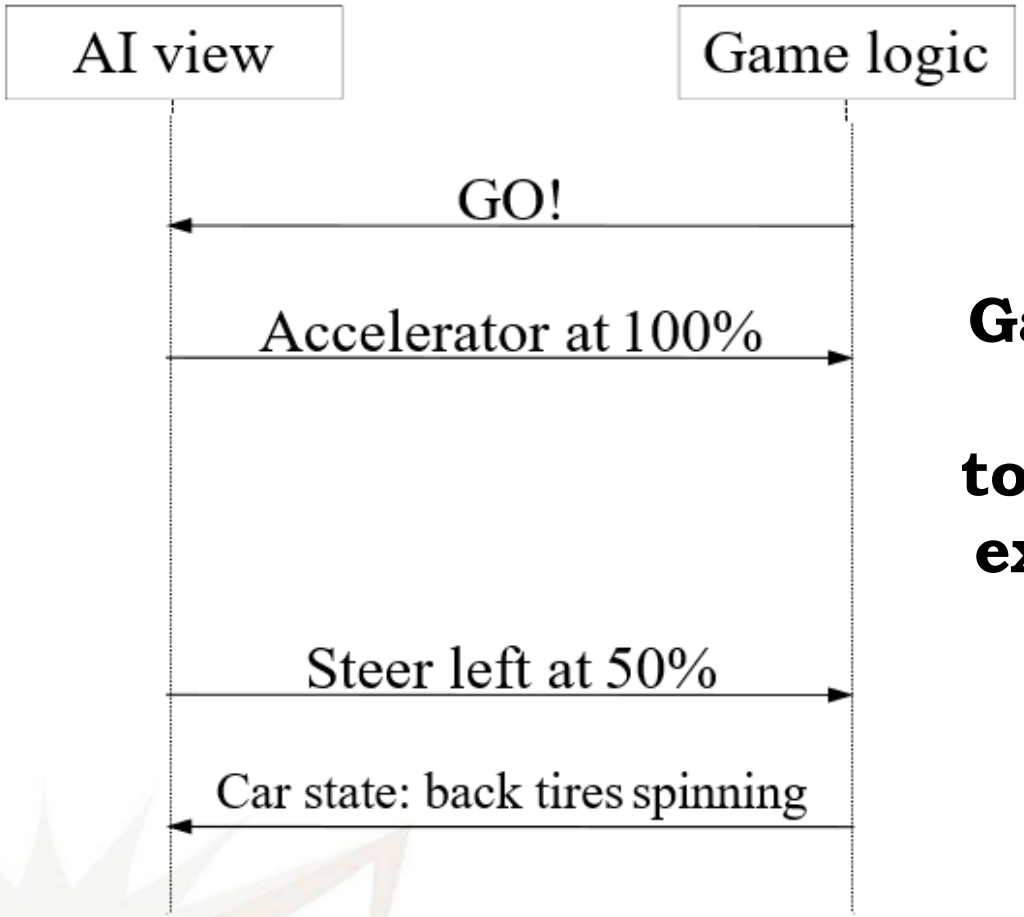
Game View

Human Player



Game View

AI Agent



Game status and game events to/from views are exactly the same

Important thing(s)…

- This is a flexible model
- Understand your game components, plan ahead before jumping to code
- Think of: game state, game rules, events and view
- Separate (1) input, (2) game logic and (3) rendering as much as possible (following the idea of separation of concerns)
- For this course, the application layer (OS+HW) is pretty much taken care of (by a game engine)

Game AI Fundamentals

What is AI

"Making computers able to perform the thinking tasks that humans and animals are capable of"

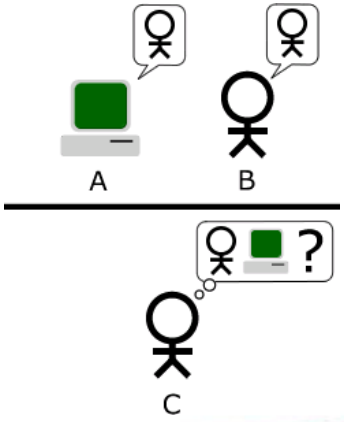
[Chapters 1-2 from AI-M]

- **Academia vs. Game Developers**
- **Academia: motivations include philosophy, psychology, engineering**
- **Game Developers: motivations are primarily engineering**

Game AI Fundamentals

Academic AI

- **The Early Days (until mid-1950's)**
  - Primarily, before computers…
  - Philosophical questionings:
    - What produces thought?
    - Could you give life to an inanimate object?
  - Creation of mechanical models exhibiting animated, animal-like behaviours
  - The Turing Test



Game AI Fundamentals

Academic AI Continued…

- **The Symbolic Era (late 1950's – early 1980's)**
  - **Symbolic systems with algorithms with 2 components**
    - **A set of knowledge (or symbols) and**
    - **A reasoning algorithm that manipulates those symbols**
  - **For example, an Expert System**
  - **Other approaches applicable to games: pathfinding, decision trees, state machines, steering algorithms**

Game AI Fundamentals

Academic AI Continued…

- **The Modern Era (1980's – 1990's)**
  - Symbolic techniques lack ability to emulate human intelligence
  - Natural computing techniques inspired by biology and other natural systems
  - E.g., Neural Networks, Genetic Algorithms, Simulated Annealing
  - Eventually researchers realized the need to be able to handle uncertainty
  - Result was modern statistical AI including Bayes nets, Support-Vector Machines

Game AI Fundamentals

Academic AI Continued…

- **Engineering**
  - **New probabilistic approach to AI became a key tool for solving real-world problems**
    - **E.g., Google's search technology**
  - **One approach is not better than others but the narrower the problem domain, the easier for an algorithm to be effective**
  - **Current trend: unified framework for symbolic and probabilistic computation**
  - **In gaming, nearly always the AI used is based on symbolic systems**

Game AI Fundamentals

Game AI (Early Development)

- PacMan (1979) - [http://www.pacmangame.net/]
- Golden Axe (1987) – not much more AI than PacMan
- Warcraft (1994) – Pathfinding
- Metal Gear Solid (1998) – Sense simulation
- Warhammer: Dark Omen (1998) – Formation motion
- Creatures (1997) – AI was the point of the game: Neural-network brain for each unit
- The Sims (2000), Black and White (2001) – AI as a main point of attraction in game
- Halo (2001) – Behavior Tree
- F.E.A.R. (2005) – Context-sensitive behaviors

Game AI Fundamentals

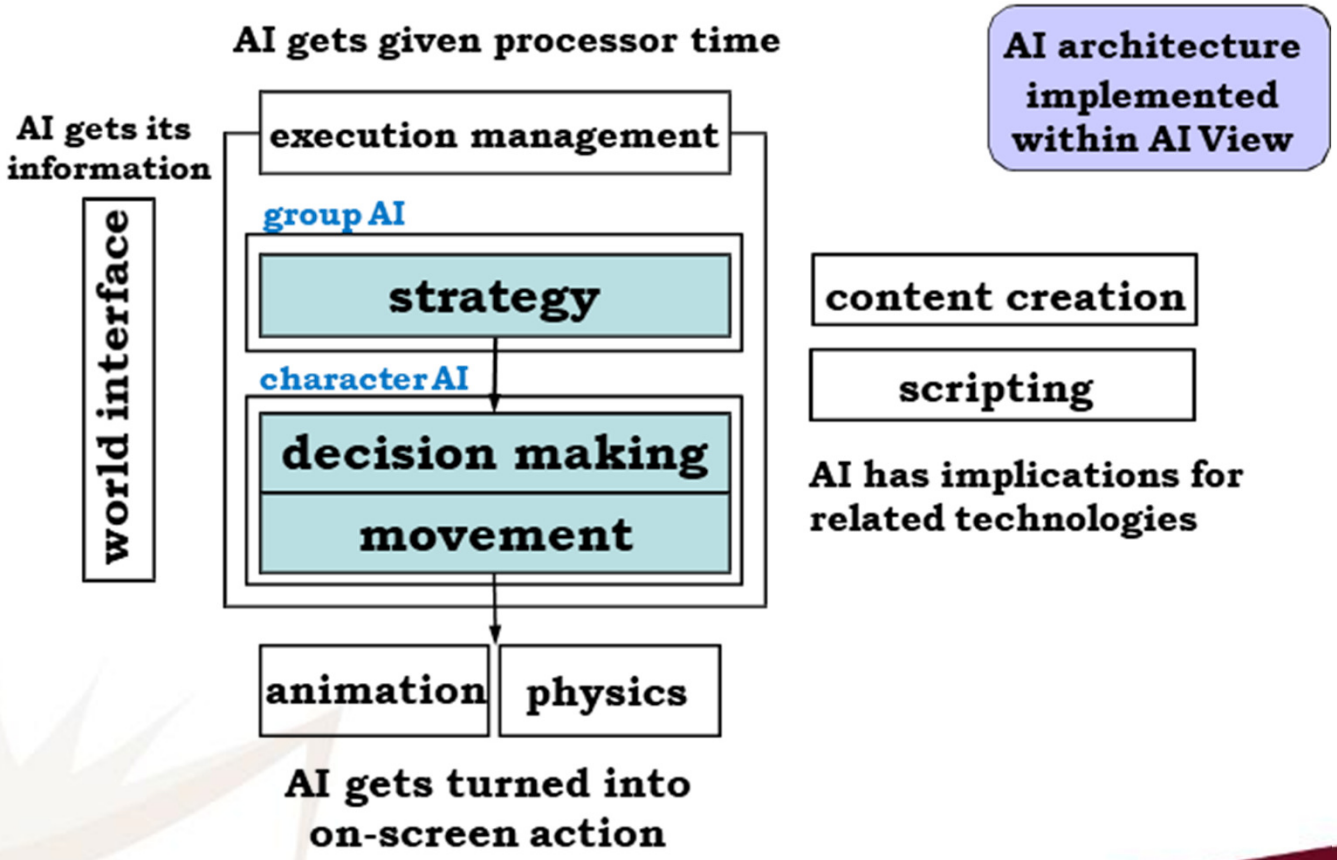
3 basic needs for Modern Game AI

1. Ability to move characters
2. Ability to make decisions about where to move
3. Ability to think tactically and strategically

Although Game AI has evolved from an almost exclusive use of state-based AI to a wide range of techniques, they all still fulfill these three basic requirements

Game AI Fundamentals

[AI-M]'s Model of Game AI



Game AI Fundamentals

AI Model: main sections

- **Decision Making**
  - Algorithms that involve a character deciding what to do next, usually from a range of available behaviours
  - E.g., animals in Zelda games stand still until a player gets too close, then they move away a small distance

  - E.g., enemies in Half Life 2 will try different complex attacks to reach the player
  - Some decisions require movement AI to carry them out (e.g., a melee (mêlée) attack)

Image from http://www.freegameaccess.com/zelda-seeds-of-darkness.htm

Game AI Fundamentals

AI Model: main sections

- **Strategy**
  - **To coordinate a team (group of characters), some strategic AI is required**
  - **Each character in the group may have their own decision making/movement algorithms, but overall their decision making will be influenced by the group strategy**
  - **E.g., enemies in Half Life work as a team to surround the player; one would often rush past the player to take up a flanking position**

Game AI Fundamentals

AI Model: Infrastructure

- AI algorithms need to work within an infrastructure such that:
  - Movement requests can be turned into action through animation/physics simulation
  - AI receives information from the game to make sensible decisions (called "perception" – what a character knows)
  - The AI system can be managed to use the right amount of CPU/memory.
- In this course we will cover each of these. Animation simulation will be covered in the lab

Game AI Fundamentals

AI Model: "Agent"-based AI

- This approach is also called an agent-based model
- i.e., producing autonomous characters, that
  - Take in information from the game data,
  - Determine what actions to take based on the information, and
  - Carry out these actions
- Essentially, a bottom-up design
- What about non-agent-based AI? Top-down design?
- [AI-M] opts not to use the term "agent", but rather considers game characters regardless of how they are implemented.

Game AI Fundamentals

Complexity Fallacy

- Good AI is about matching the right behaviours to the right algorithms

1. When Simple Things Look Good
   - E.g., PacMan: the balance of simple AI and a variety of AI creates great gameplay – even creating impression of even more complex AI
2. When Complex Things Look Bad
   - The latest, hyped AI isn't always the best to use
   - Knowing when to be complex and when to be simple is a tricky element of AI programming to learn

Game AI Fundamentals

Complexity Fallacy continued…

3. **The perception window**
   - **Typically a player will only interact with a character for a short time.**
   - **There is not enough time to fully understand a character's situation**
   - **A character's AI must match its purpose in the game and the attention it will get from the player**
4. **Changes of behavior**
   - **Changes in behaviour in a character should be predictable (e.g., when the player is spotted)**

Game AI Fundamentals

The Kinds of AI in Games

- **Hacks**
  - **aka Behaviourism. We are not particularly interested in the nature of reality or mind: we want characters that look right.**
  - **You start with a design for a character and apply the most relevant tool to get the result (even if it is not a recognizable AI technique)**

Game AI Fundamentals

The Kinds of AI in Games

- **Heuristics**
  - Common heuristics: Most Constrained, Do The Most Difficult Thing First, Try Most Promising Thing First
  - Probably will not work in all situations
  - When sacrificing accuracy, a heuristic is used
  - Frequently, specific to a game
- **Algorithms**
  - General bits of AI, such as movement decision making, and tactical thinking all benefit from tried and tested methods that can be reused
  - These are the focus of [AI-M]

Game AI Fundamentals

AI Engine

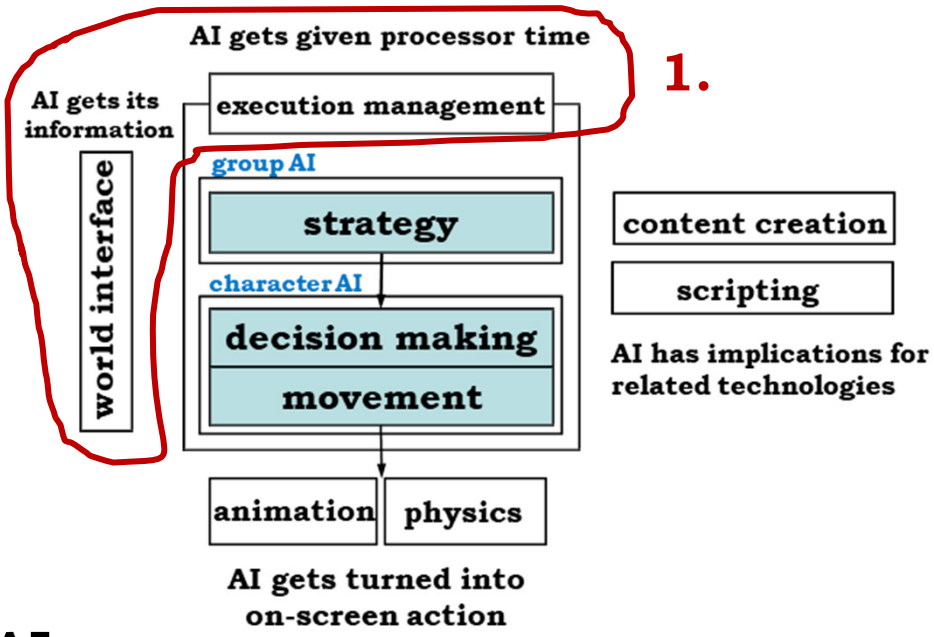
- A change in the way that games are developed in the last 20 years
- Initial style: Bits of AI code intermingled with other game code, AI written for each game and each character, each character's behaviour controlled by a small program
- Now: AI Engine
  - Building general components that can be reused and combined in appropriate ways
  - Allows characters to be designed by level editors/ artists/players
  - Integrated as part of a Game Engine

Game AI Fundamentals

Structure of an AI Engine

Two main components:

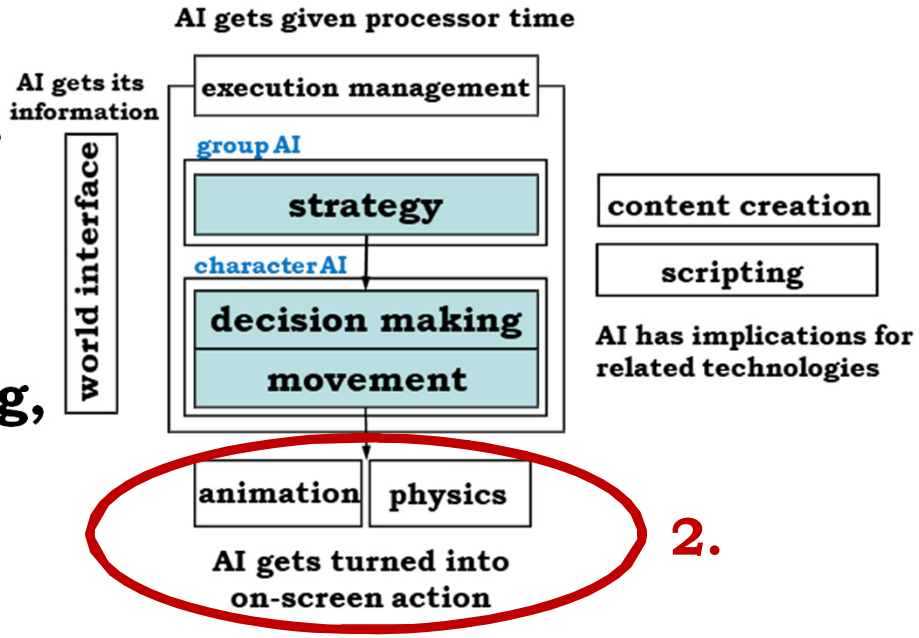
1. **Infrastructure in two categories:**
   - **A general mechanism for managing AI behaviours (decide what gets run, etc.)**
   - **A world-interfacing system for getting information into the AI**



Game AI Fundamentals

Structure of an AI Engine

2. **Infrastructure in two categories:**
   - **Whatever the AI wants to do should translate information to screen action (normally, movement, pathfinding, etc.) or background computation (tactics, strategies to win)**

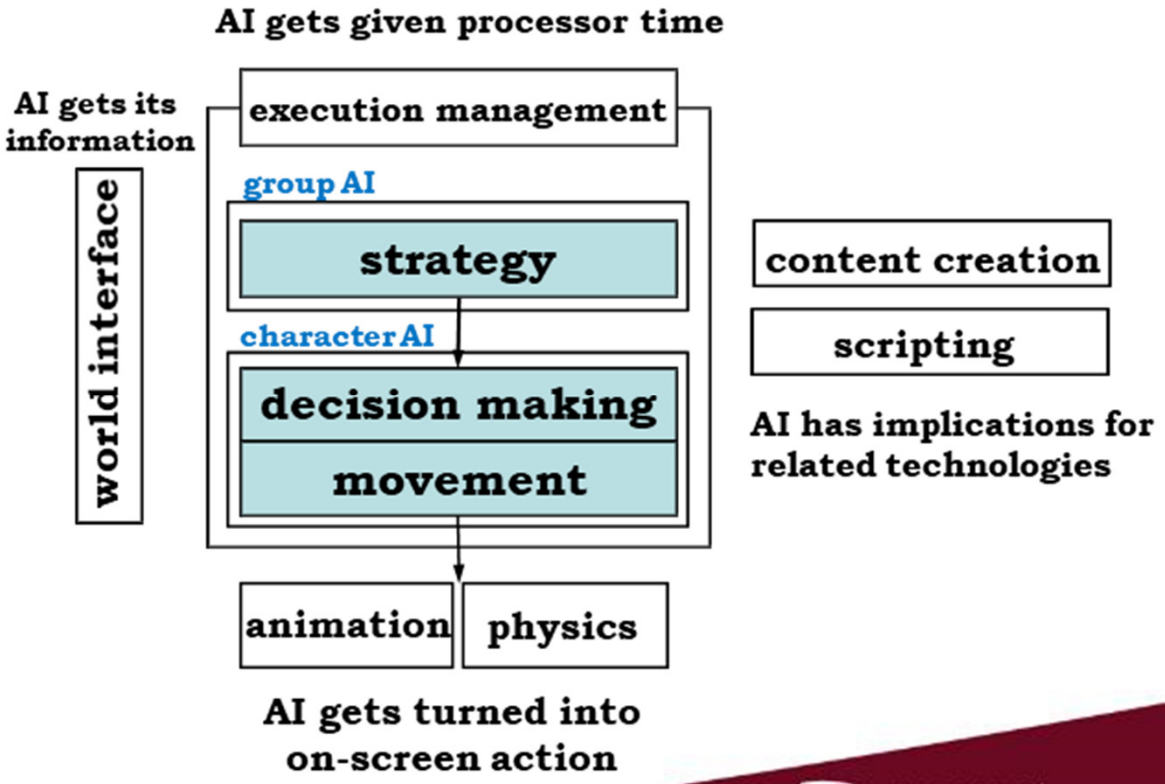


Game AI Fundamentals

Structure of an AI Engine

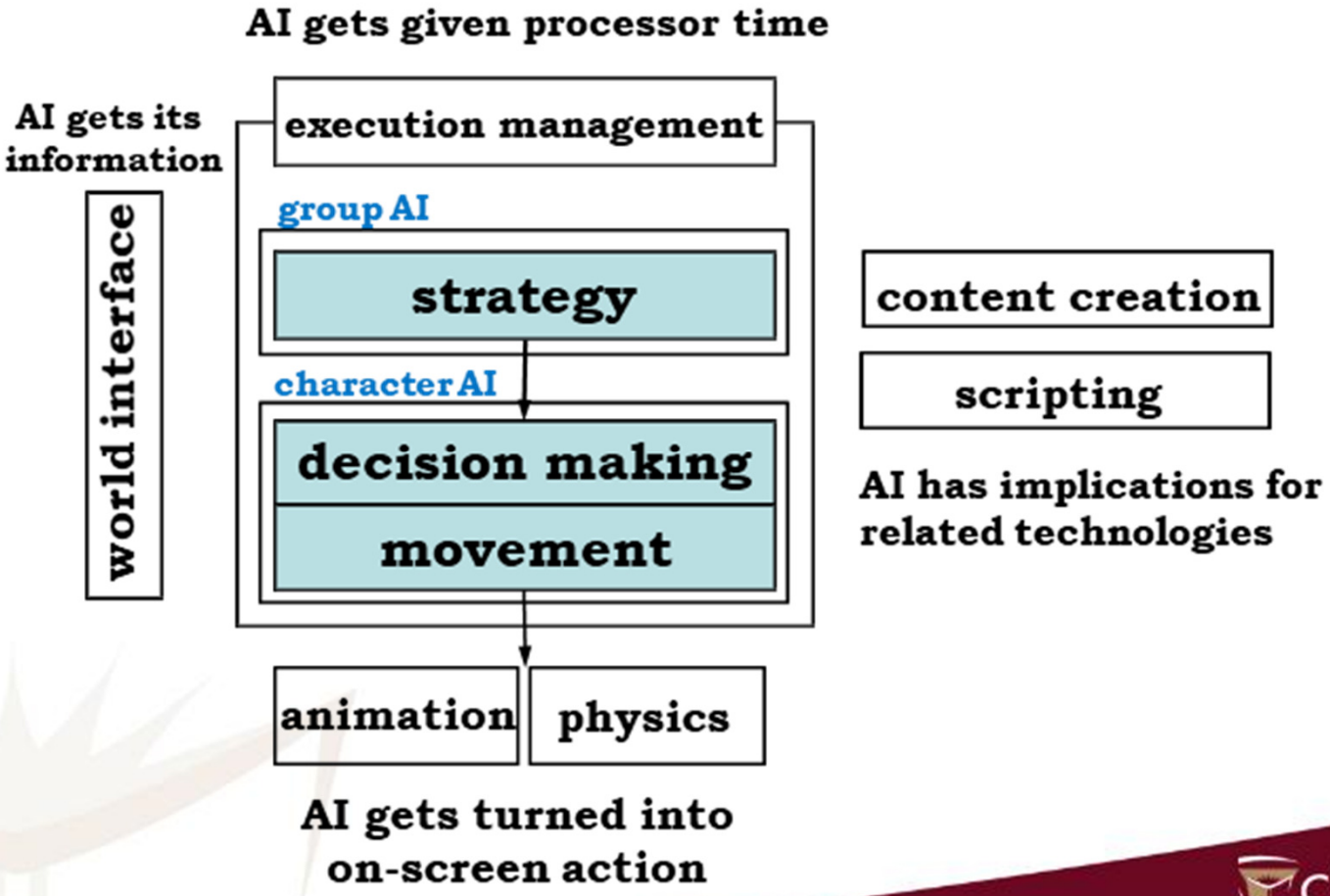
Standard behaviour structure that liaisons between the two components

- **Having all AI conform to the same structure for easy development of future code**



Movement AI

Movement AI in [AI-M]'s Model



Movement AI

Movement AI vs. Animation

- Movement forms the lowest level of AI techniques in the AI Model
  - Many games rely solely on some movement AI, and very little advanced decision-making
  - Some games have no movement AI (e.g., turn based games such as Chess)
- There is some degree of overlap between Movement AI vs. Animation (which is also movement)
  - We will focus on large-scale Movement AI – the movement of characters around the game level and NOT the movement of limbs/faces/parts etc.

Movement AI

Basics of Movement Algorithms

- All movement algorithms have the same basic form:
  - **Input:** They take geometric data from their own state and the **state of their environment,** and
  - **Output:** They come up with a geometric output **representing the movement** they would like to make
- Movement algorithms assume Characters as Points
- Inputs range
  - from simply the positions of the character and the enemy,
  - to a lot of interaction with the game state and level geometry (e.g., to avoid bumping into walls)

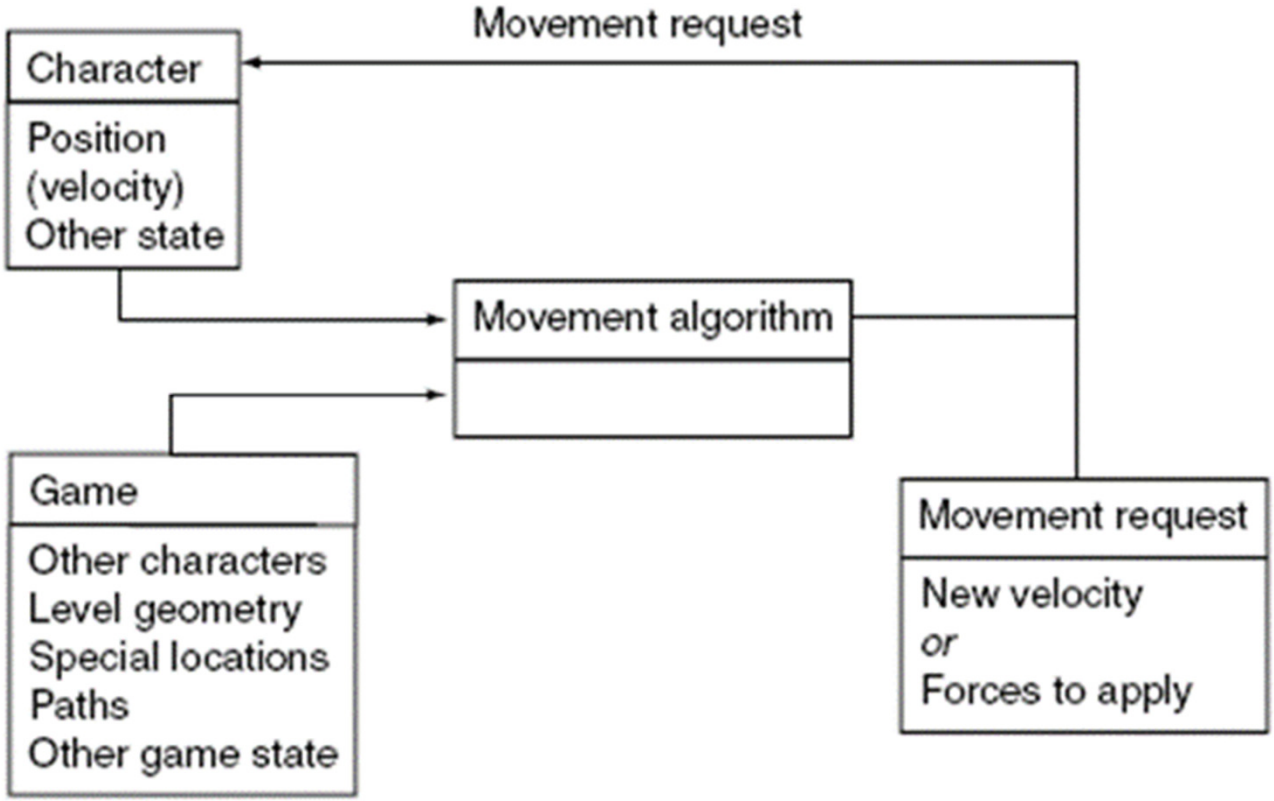
Movement AI

Basics of Movement Algorithms

- All Similarly, the output can vary.
- One type of output is kinematic movement. E.g., outputs simply a *direction and velocity* to move in. The movement does not account for how characters accelerate and slow down.
- Another type is dynamic movement (aka "steering" behaviours). Such movement takes into account the current motion (e.g., current velocities) of the character. Outputs forces or accelerations with the aim of changing the velocity of the character. E.g., to reach a destination, first accelerate towards it, then decelerate so to stop at the destination.

Movement AI

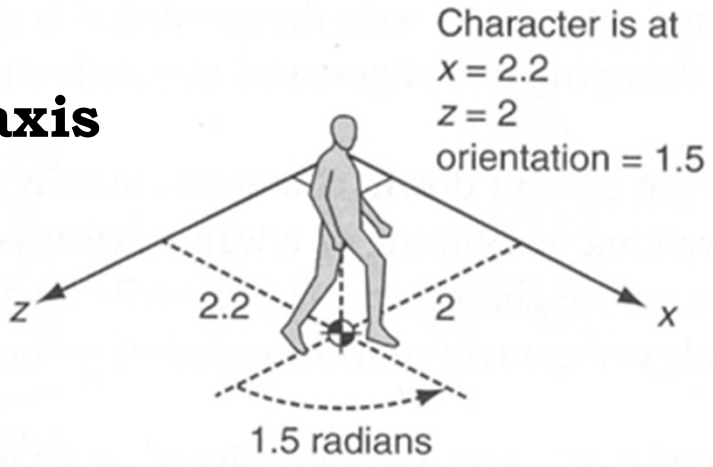
Movement algorithm structure



Movement AI

2D Movement - Statics

- Static structure
  - Position of character (two linear coordinates in a 2D vector)
  - Orientation (floating point value in radians)
- 2D movement commonly take place on the x-z plane with y axis ("up" direction) fixed to zero
- In standard game engines, by default, a character is looking down the z-axis at zero orientation



Movement AI

2½ Dimensions

Definition as per [AI-M]

Note: This is not the common definition!

- Since some of the math in 3D geometry is complicated, frequently developers use a hybrid of 2D and 3D geometry known as 2½D, or four degrees of freedom
- In 2½D, we use 3D position, but represent orientation as a single value (as in 2D).
- This assumption is reasonable when a character is constrained to the ground while remaining upright. Then the only component of the orientation is the rotation about the "up" direction.
- 3D positioning still allows for jumping, dropping from ledges, using elevators, etc.

Movement AI

[AI-M]'s coding (data structures)

- **Kinematic structure**
  - **position** of character (2D or 3D vector)
  - **orientation** (floating point value indicating degree of facing)
  - **velocity** (2D or 3D vector indicating speed and direction)
  - **rotation** (floating point value indicating rotation speed, aka angular velocity)
- **Steering structure**
  - **Accelerations: linear** (for linear acceleration)
    angular (for rotational accel.)

Movement AI

Updating Position

- **High-school physics equations for motion (with t being the duration since the last update):**

$$s = s + vt + 0.5at^2$$

Where (bolded) $s = (s_x, s_y)$ or $s = (s_x, s_y, s_z)$, etc.

- **If frame rate high, the duration between updates is small, the square is even smaller, contribution of acceleration is negligible**
- **Common to see the 2nd term removed from the update loop for fast movement updates (less computation, too)**

$$s = s + vt$$

- **Also: $v = v + at$**

Movement AI

Updating Orientation

- **Similarly for orientation:**

$$\Theta = \Theta + \omega t$$

with ω as angular velocity and

$$\omega = \omega + \alpha t$$

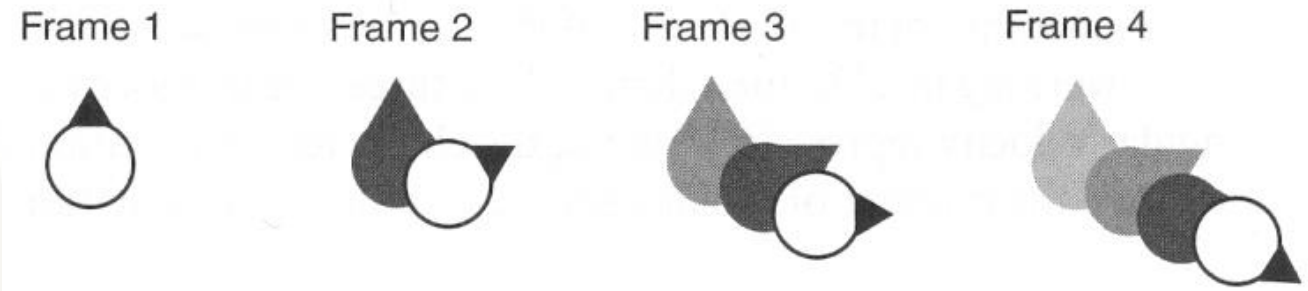
with α as angular acceleration

- **E.g., to compute the next orientation, the angular velocity is computed first using $\omega = \omega + \alpha t$, and then the orientation $\Theta = \Theta + \omega t$ using the updated ω**
- **Known as Newton-Euler-1 integration**

Movement AI

Independent Facing

- Nothing connects the direction of a character's motion with the direction it is facing.
- So, characters should orientate themselves to face the direction they are traveling.
- Doing this abruptly should be avoided; better to change the orientation, for example, a portion of the way the character should be facing each frame



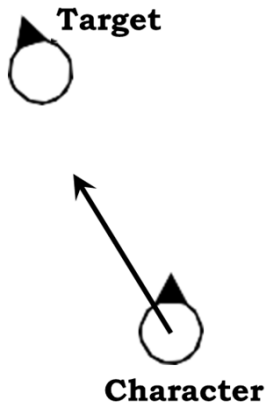
Movement AI

Kinematic Movement Algorithms

- Movement (as a desired velocity) is calculated using static data (position & orientation; no velocities or acceleration)
- No acceleration involved? A little unrealistic, but works fine for many games. Simpler to implement.
- Abrupt changes in velocity can be smoothed over several frames to give realistic look
- "Kinematic movement algorithms still form the bread and butter of movement systems in most games" [AI-M]

Movement AI

Seek (Kinematic)



- Input: Character's and Target's static data (position & orientation)
- Calculates:
  - Velocity direction, a vector, from the character to the target:

    position(target) – position(character) or

    $p_t - p_c$
  - No rotation
- Perform normalization on the direction vector to obtain unit vector, which will then be multiplied with the maximum velocity v_m of the character

Movement AI

Seek (Kinematic) - Example

- **Input: character has position (11,2) and velocity (0,3); target has position (4,9) and an unknown velocity. Time between updates is 1/4. Maximum velocity is 4, and maximum acceleration is 4.**
- **Next position calculation:**
  - **Velocity direction = p_t - p_c = (4, 9) – (11, 2) = (-7, 7)**
  - **Normalized velocity direction =**

$$\frac{p_t - p_c}{|p_t - p_c|} = \frac{1}{\sqrt{7^2 + 7^2}}(-7, 7) = \frac{1}{\sqrt{2}}(-1, 1)$$

  - **Kinematic Seek velocity =**

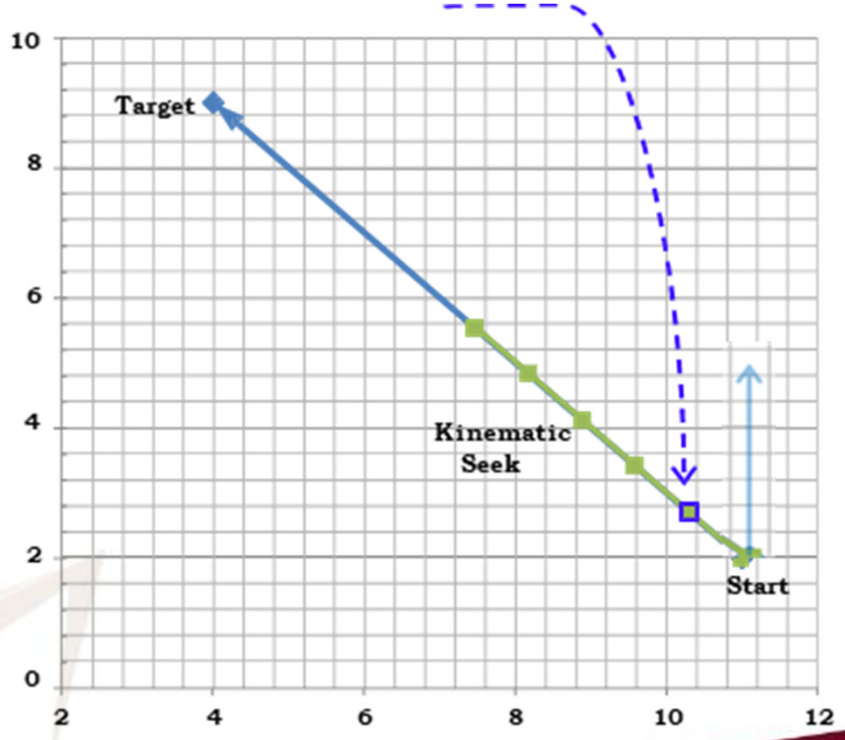
$$v = v_m\left(\frac{p_t - p_c}{|p_t - p_c|}\right) = 4\left(\frac{1}{\sqrt{2}}(-1, 1)\right)$$

Movement AI

Seek (Kinematic) - Example

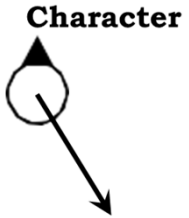
- **Next position =** $p'_c = p_c + v_t = (11,2) + 4\left(\frac{1}{\sqrt{2}}(-1,1)\right)\left(\frac{1}{4}\right)$

= (10.293, 2.707)



Movement AI

Flee (Kinematic)



- Simply reverse the calculation of the velocity direction vector, to move away from the target
  - Calculate from target to character:
    position(character) – position(target)
    or $p_c - p_t$
- Again, normalize this direction vector and multiply it by the maximum velocity v_m of the character:

$$v = v_m \left(\frac{p_c - pt}{|p_c - pt|} \right)$$

Movement AI

Arrive (Kinematic)

- Seek is designed for chasing
- If the character seeks a particular location in the world at constant maximum speed, it is likely to overshoot an exact point, wiggle backward and forward trying to get there.

Target

Character

Movement AI

Arrive (Kinematic)

- **To avoid this problem, Kinematic Arrive introduces**

  I. **A radius of satisfaction,** r_{sat}**, (to check if the character is nearing location):**

  If $|p_t - p_c| > r_{sat}$ **then** $|v| = v_m$ **else** $|v| = 0$ **or**

  II. **A time-to-target value,** $t2t$**, (to slow character down) using**

  $$|v| = \min(v_m, |p_t - p_c|/t2t)$$

  - **This reduces as the character is nearing location**
  - **If this is faster than the maximum speed,** v_m**, we use** v_m **instead as the character's velocity**

Movement AI

Arrive (Kinematic)

- **Or blend the approaches I and II, such that outside the (a much smaller) satisfaction radius,**

  $r'_{sat} < r_{sat}$, use time-to-target:

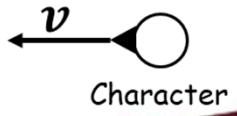
  If $|p_t - p_c| > r'_{sat}$ **then**

  $$\boldsymbol{v} = \min(v_m, |\boldsymbol{p}_t - \boldsymbol{p}_c|/t2t)$$

  // v is the magnitude of the vector v from
  // the character to the target

  else

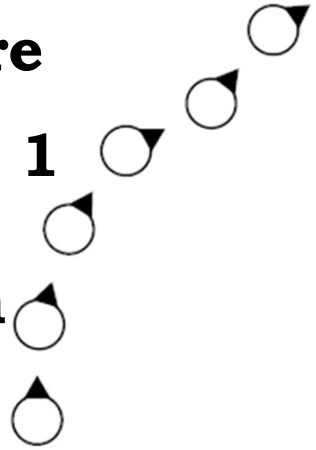
  $|v| = 0$



Movement AI

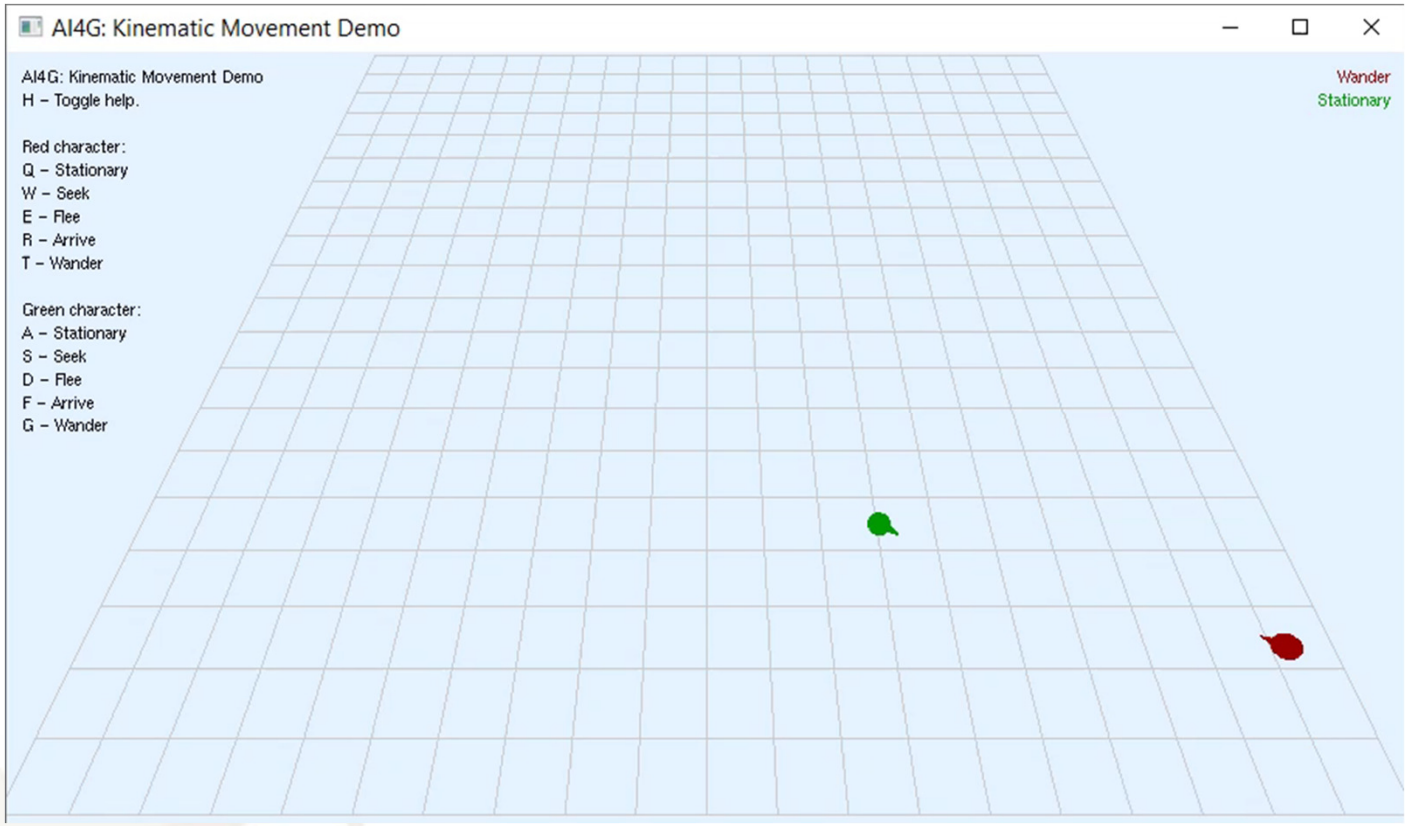
Wander (Kinematic)

- **Character meanders randomly (like a random walk, but smoother) in a forward direction**
- **Always moves in the direction of the current orientation at maximum velocity v_m**
- **Direction of orientation is randomized here**
  - **Take a random number between –1 and 1 (where values around zero are more likely), and multiply by fixed maximum rotational speed to get new rotation ω**
  - **Then $\Theta = \Theta + \omega t$ for new orientation**
- **DEMO**



Movement AI

Wander (Kinematic)



https://github.com/idmillington/aicore

Movement AI

Steering Behaviours

- Use velocity ***and acceleration*** to determine movement.
- Extend kinematic algorithms by adding velocity and rotation as input – thus characters have acceleration
- 2 categories:
  - Fundamental steering behaviours
  - Behaviours built from a ***combination*** of fundamental behaviours

Movement AI

Variable Matching

- Simplest form of steering behaviours involve matching variables from character with variables from target.
- Matching
  - Position of target (Seek, Arrive)
  - Orientation of target (Align)
  - Velocity of target (Velocity Matching)
  - Delegation or Combination of various kinematic elements
- There could be opposite behaviours that will intend to “unmatch” as much as possible.

Movement AI

Seek & Flee

- **Seek** – tries to match the position of the character with the position of the target.
- To do this, maximum acceleration is applied to the direction to the target.
- Over time, velocity keeps increasing, since acceleration will cause its speed to grow.
- **Post-processing:**
  - Position, velocity, orientation, rotation are updated after steering behaviour finishes
  - Velocity/speed of character needs to be clipped (bounded) so to not exceed its maximum value

Movement AI

Seek (Steering) - Example

- **Input: character has position (11,2) and velocity (0,3); target has position (4,9) and an unknown velocity. Time between updates is 1/4. Maximum velocity is 4, and maximum acceleration is 4.**
- **Next position calculation:**
  - **Steering Seek acceleration =**

$$\boldsymbol{a} = a_m\left(\frac{\boldsymbol{p}_t - \boldsymbol{p}_c}{|\boldsymbol{p}_t - \boldsymbol{p}_c|}\right) = 4\left(\frac{1}{\sqrt{2}}(-1,1)\right)$$

  - **Steering Seek velocity =**

$$\boldsymbol{v} = \boldsymbol{v}_c + \boldsymbol{a}t = (0,3) + 4\left(\frac{1}{\sqrt{2}}(-1,1)\right)\left(\frac{1}{4}\right)$$

$$= (-0.707, 3.707)$$

Note: $|v| = 3.774 < 4$**, so OK.**

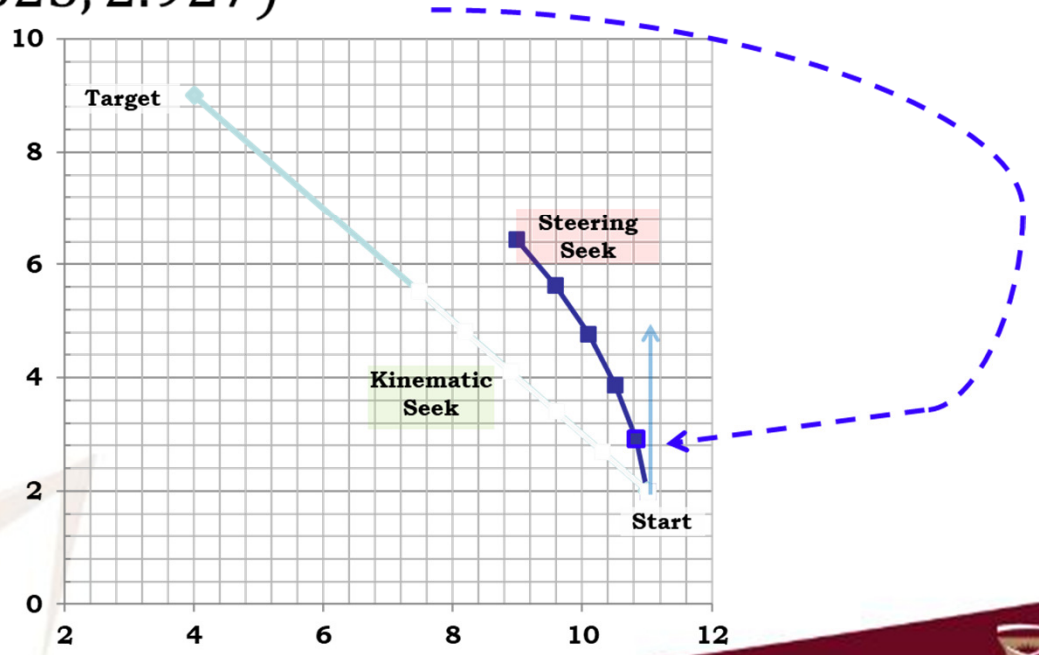
Movement AI

Seek (Steering) - Example

- **Next position =**

$$\boldsymbol{p}'_c = \boldsymbol{p}_c + \boldsymbol{v}t = (11,2) + (-0.707, 3.707)\left(\frac{1}{4}\right)$$

$$= (10.823, 2.927)$$



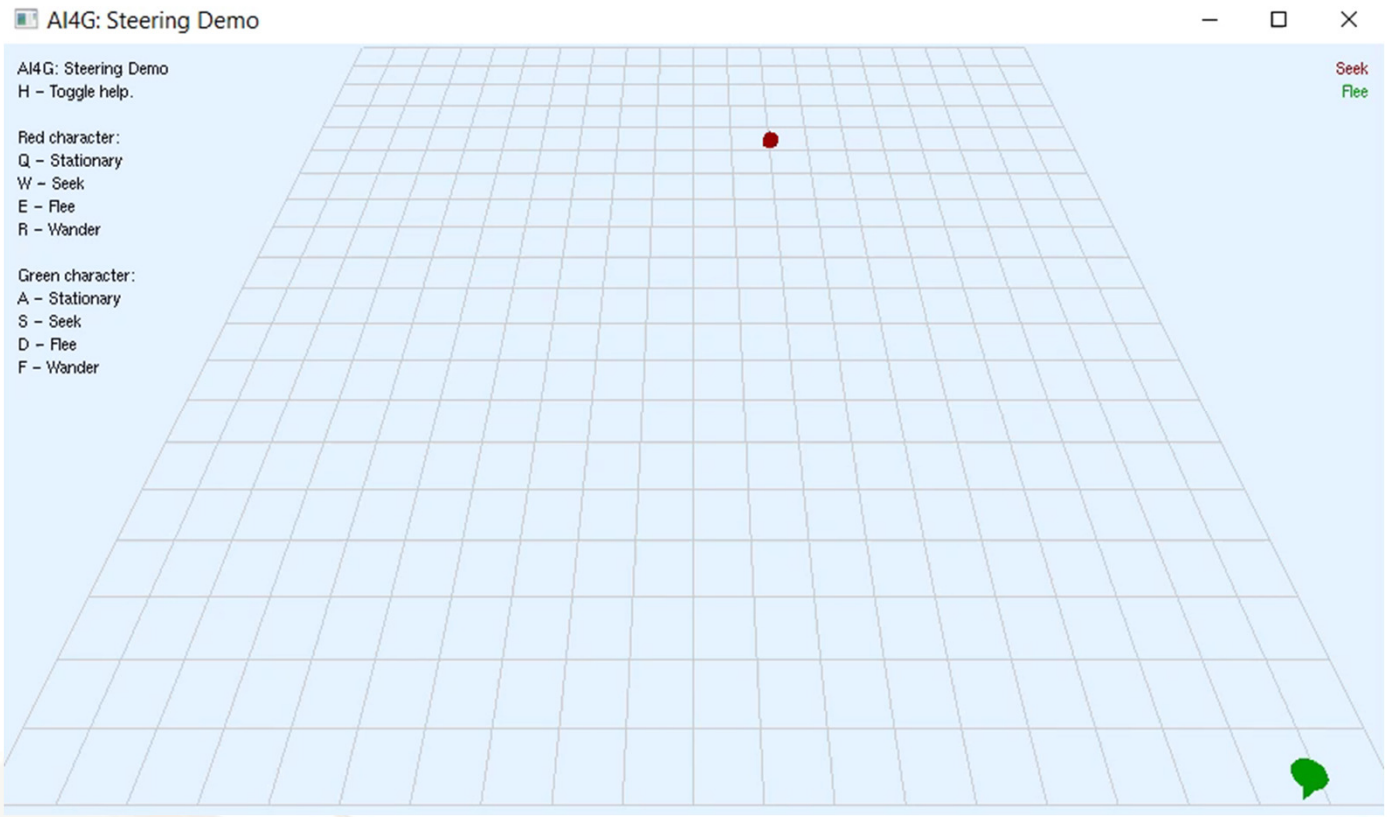
Movement AI

Seek & Flee (Steering)

- Since the speed for Seek cannot grow unbounded, alternatively, we can limit the speed by introducing additional drag to slow down the character a little at each frame.
  - Drag also prevents the orbiting of a moving target, the orbit becomes an inward spiral
- Flee – Direction of maximum acceleration from target to character (so it accelerates away)
- No change in orientation for both Seek and Flee
- DEMO

Movement AI

Seek & Flee (Steering)

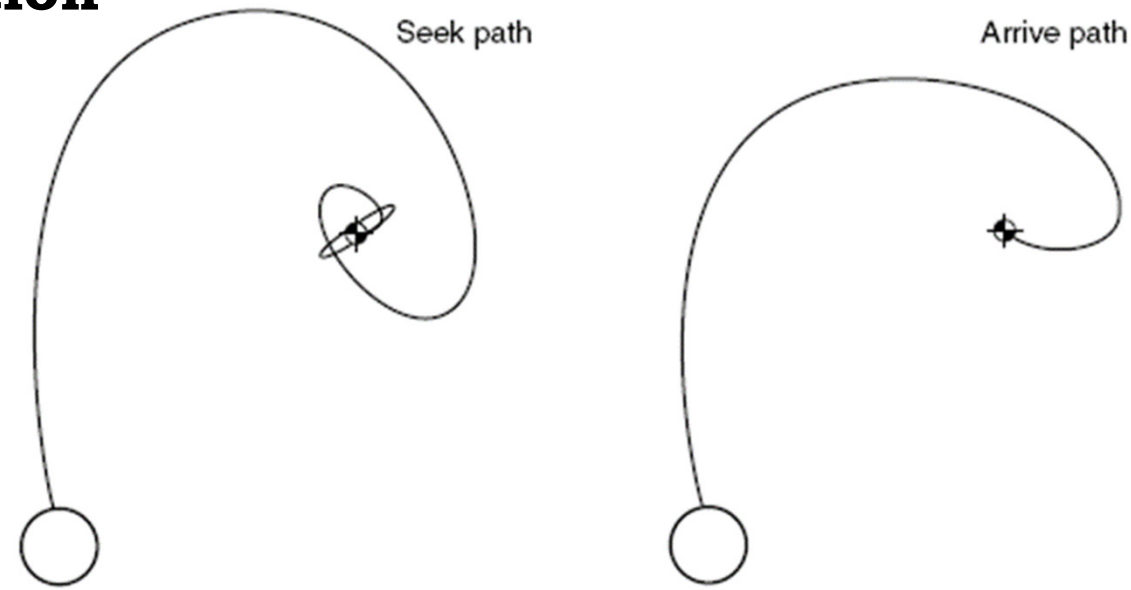


https://github.com/idmillington/aicore

Movement AI

Arrive (Steering)

- **Similar to (Kinematic) Arrive, this (Dynamic) Arrive intends to slow the character down as it approaches the target so that it arrives exactly at the right location**



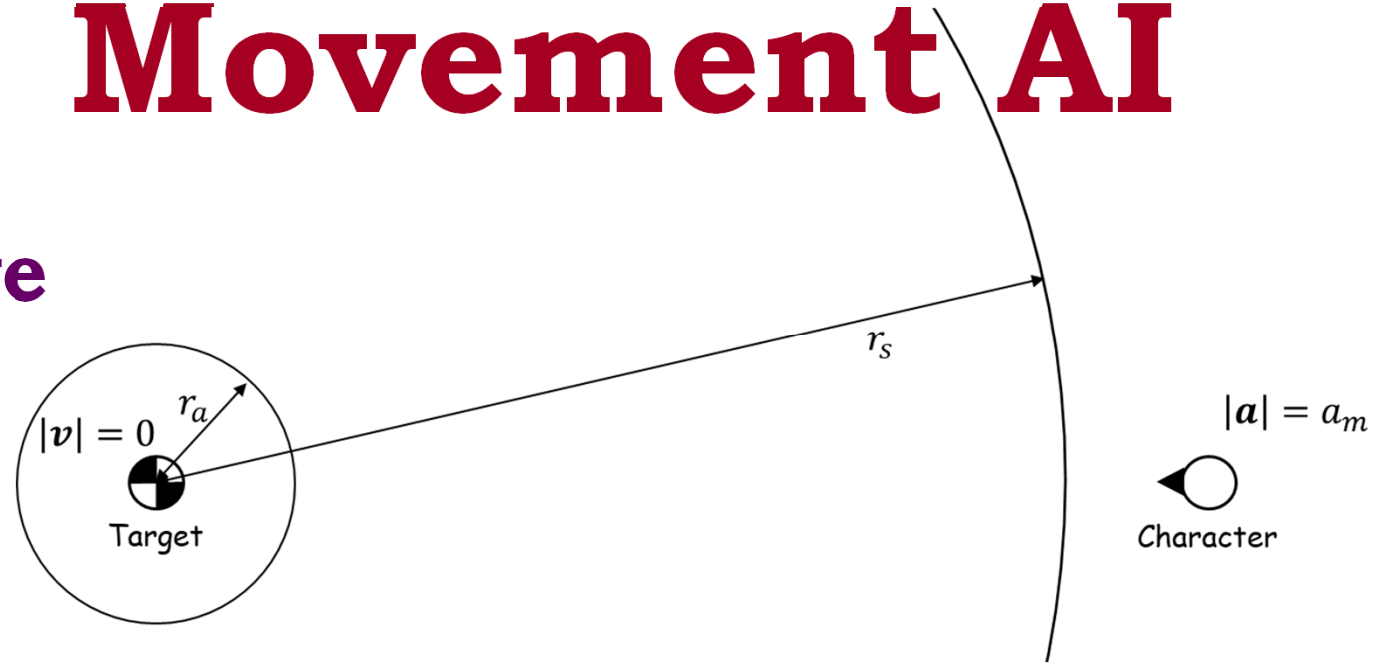
Movement AI

Arrive

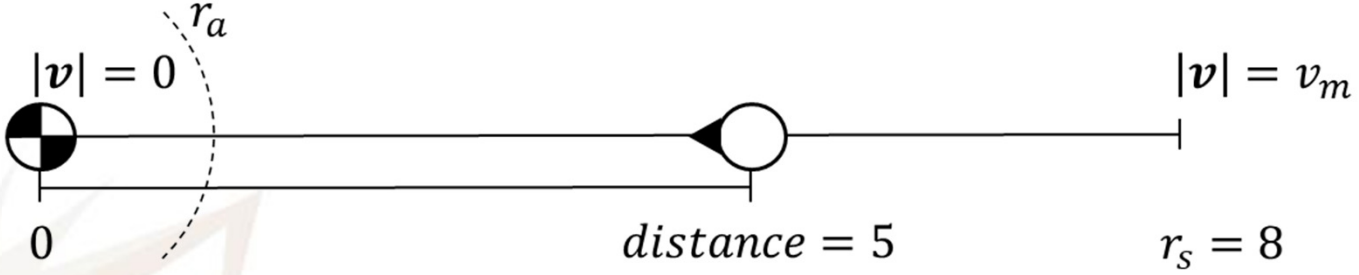
- Uses 2 radii
  - Arrival Radius (as with the satisfaction radius in Kinematic Arrive) r_a – lets character get near enough to target w/o letting small errors keeping it in motion
  - Slowdown Radius (larger) r_s – slows character down when it passes into this radius. Ideal velocity (goal velocity) is calculated using time-to-target method (like before). Upon entering this radius, velocity is maximum, v_m. When it arrives successfully (within the arrival radius r_a), the velocity is zero.

Movement AI

Arrive



- The goal velocity (speed) is interpolated between maximum (at slowdown radius) and zero with respect to the distance to target



- Here the goal velocity is: $|\boldsymbol{v}| = \left(\frac{5}{8}\right) v_m$

Movement AI

Arrive

- Since Arrive is a steering behavior we can't just assign the goal velocity as the character's velocity, but we have to use an acceleration to adjust the character's velocity to be closer to the goal velocity.
- The acceleration of the character is then
  (goal velocity - actual character velocity)/
  (time-to-target)
- When a character is moving too fast to arrive at right time, i.e.,
  goal velocity < actual character velocity,
  acceleration will be negative, or slowing it down
- Large accelerations still capped at max value, a_m

Movement AI

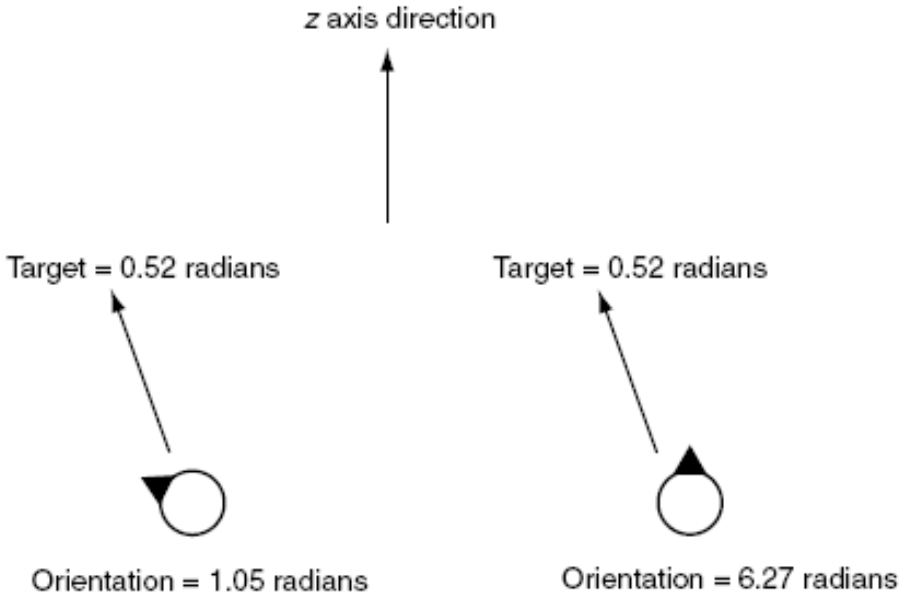
Leave

- Opposite behaviour to Arrive
- No point in implementing – unlikely to want to accelerate and build up speed if leaving.
- Just use Flee (move away at maximum acceleration)

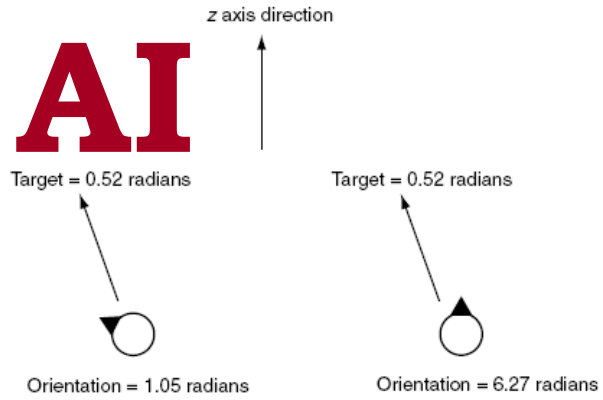
Movement AI

Align

- Match orientation of the character with that of target
- Pays no attention to position/velocity of character/target
- Idea: Subtract character orientation from target orientation, and convert result into range (-π, π) radians
- Algorithm is similar to Steering Arrive (with target and slow radii, etc.)



Movement AI

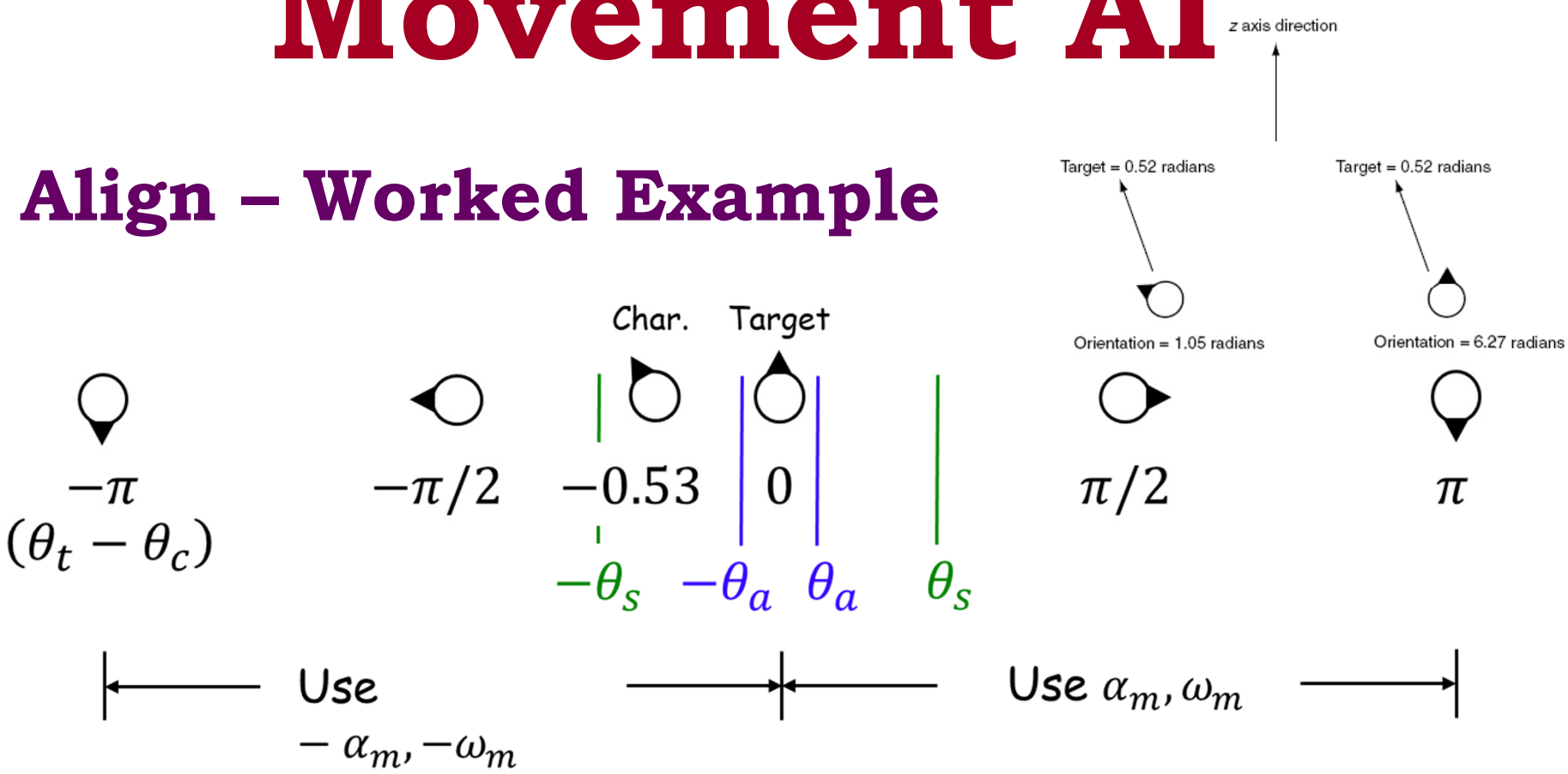


Align – Worked Example

- If orientation of target is θ_t = 0.52 radians and the character orientation is θ_c = 1.05 radians, then $\theta_t - \theta_c$ = −0.53 radians (~ −30.37°).
- Assume:
  - the time step t = 0.1,
  - character's angular velocity ω_c = −1 rad/s,
  - maximum angular velocity ω_m = ±π rad/s and
  - maximum angular acceleration α_m = ±2π rad/s^2,
  - an ***arrive orientation*** of ±θ_a = ±0.05 radians,
  - a ***slow-down orientation*** of ± θ_s = ±π/4 radians, &
  - a "time to target" of $t2t$ = 0.5 seconds

Movement AI

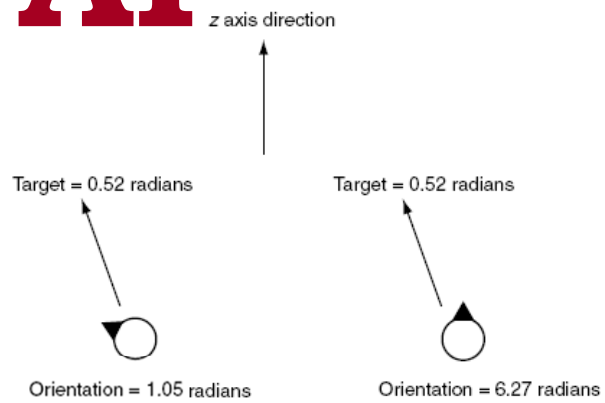
Align – Worked Example



- **Since $|\theta_t - \theta_c| = 0.53 > \theta_a = 0.05$, we need to compute an angular acceleration.**
- **This angular acceleration will be negative – Why?**

Movement AI

Align – Worked Example



- **First, the goal angular velocity is**

$$\omega_g = \omega_m \left(\frac{-0.53}{-\pi/4}\right) = (-\pi)\left(\frac{-0.53}{-\pi/4}\right) = -2.12/s$$

- **Then the angular acceleration is**

$$\alpha_c = \frac{(\omega_g - \omega_c)}{t2t} = \frac{(-2.12-(-1))}{0.5}\frac{\text{rad}}{s^2} = -2.24\frac{\text{rad}}{s^2}$$

- **Since $|\alpha_c| < |\alpha_m|$, we can then compute**

$$\omega_c = \omega_c + \alpha_c\, t = -1 + (-2.24)(0.1) = -1.224 \text{ rad}/s$$

- **Since $|\omega_c| < |\omega_m|$, we can then compute**

$$\theta_c = \theta_c + \omega t = 1.05 + (-1.224)(0.1) = 0.9276 \text{ rad}$$

- **Note that now $\theta_t - \theta_c = -0.4076$ radians**

Movement AI

Velocity Matching

- **Idea: Use acceleration to get to the target velocity**
  - **Subtract velocity of character from velocity of target to get velocity difference ($\Delta v = v_t - v_c$)**
  - **Use time-to-target method to find acceleration/deceleration ($a = \Delta v / t2t$; capped at a_m) to be applied to character to gradually achieve the velocity of the target**
- **How is matching velocities useful?**
  - **Becomes much more useful when combined with other behaviours (e.g. flocking steering behaviour)**