

Combining Behaviors

- ❑ A moving character – usually needs more than one steering behaviour to model it **more realistically**
- ❑ E.g. To seek its goal, avoid collision with others, avoid bumping into walls (all at the same time)
- ❑ Some special behaviours may require more than one steering behaviour to be active at once.
- ❑ E.g., to steer in a group towards a goal, maintaining a good separation distance from group members, and to match each members' velocities.
- ❑ **How?**

Combining Behaviors

❑ Blending

- Execute all steering behaviours and combining their results using some set of weights or priorities

❑ Arbitration

- Selects one or more steering behaviours to have complete control over character. Many schemes available nowadays.

- ❑ Many steering systems combine elements of both blending and arbitration to maximize advantages

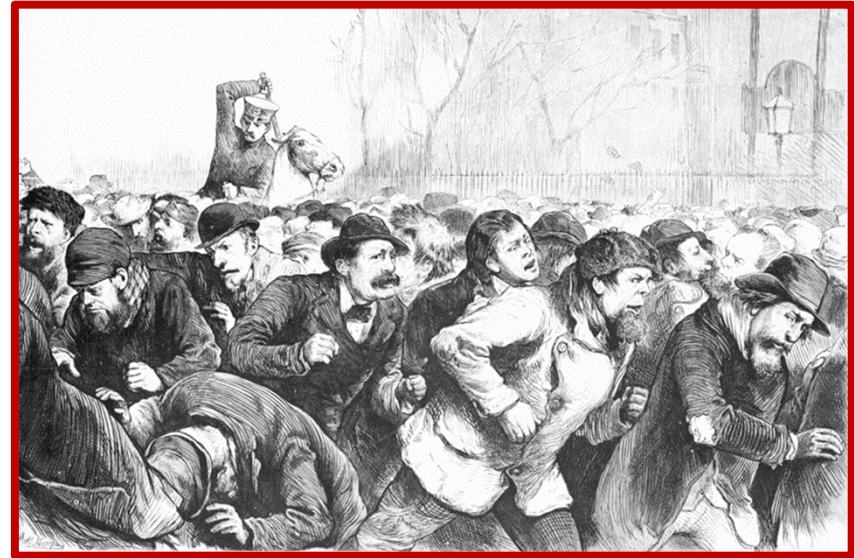
Weighted Blending

- Use **weights** to combine steering behaviours

- Example: Riot crowd AI

- In this example, the characters need to

- move as a mass (staying close to the center of mass of the group), while
- making sure that they aren't consistently bumping into each other.



- Character does not just do one thing. It **does a “blend” or synthesis of all considered behaviours**

Weighted Blending

□ Idea:

- Each steering behaviour is asked for its acceleration request, a_i
- Combine the accelerations using a weighted linear sum, weights w_i specific to each behavior,

$$a = w_1 a_1 + w_2 a_2 + \dots + w_n a_n$$

- If magnitude of a , $|a|$, from sum is too great, trim it accordingly: $a = \min(a_m, |a|) \left(\frac{a}{|a|}\right)$

- There are *no constraints on the weights* – they *don't have to sum to one*.
- The group of steering behaviours are blended to act as a single behavior.

Weighted Blending

□ Idea:

- Each steering behaviour is asked for its acceleration request, a_i
- Combine the accelerations using a weighted linear sum, weights w_i specific to each behavior,

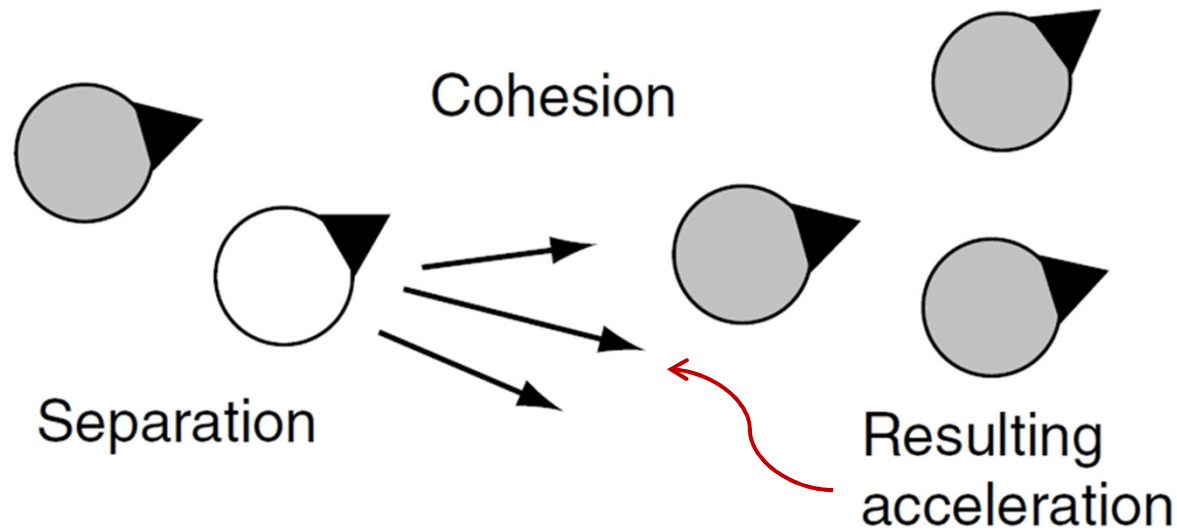
$$a = w_1 a_1 + w_2 a_2 + \dots + w_n a_n$$

- If magnitude of a , $|a|$, from sum is too great, trim it accordingly: $a = \min(a_m, |a|) \left(\frac{a}{|a|}\right)$

- There are *no constraints on the weights* – they *don't have to sum to one*.
- The group of steering behaviours are blended to act as a single behavior.

Weighted Blending

- In our riot crowd example, we may use weights of $w_i = 1$ for *both separation and cohesion*.
- In this case the requested accelerations are summed and cropped to the maximum possible acceleration. This is the output of the algorithm

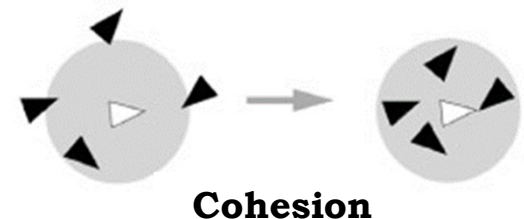
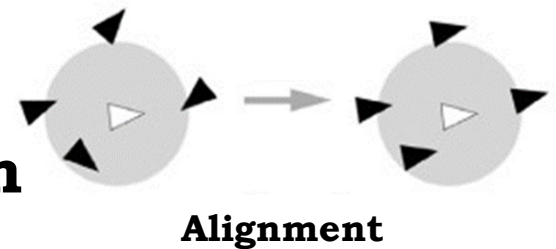
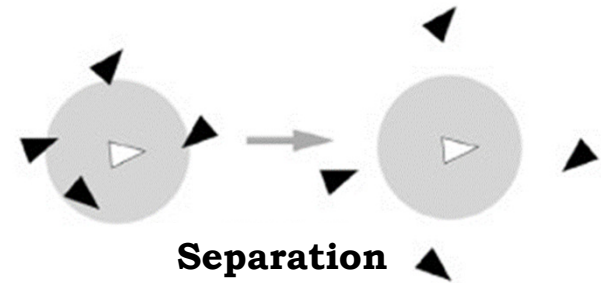


Weighted Blending

- ❑ How to choose appropriate weights for multiple behaviours?
 - As in all parameterized systems, the choice of weights needs to be the subject of inspired guesswork, or good trial and error.
 - Could also try to use dynamic weights, but autonomous systems (using, for example, machine learning) don't work well.

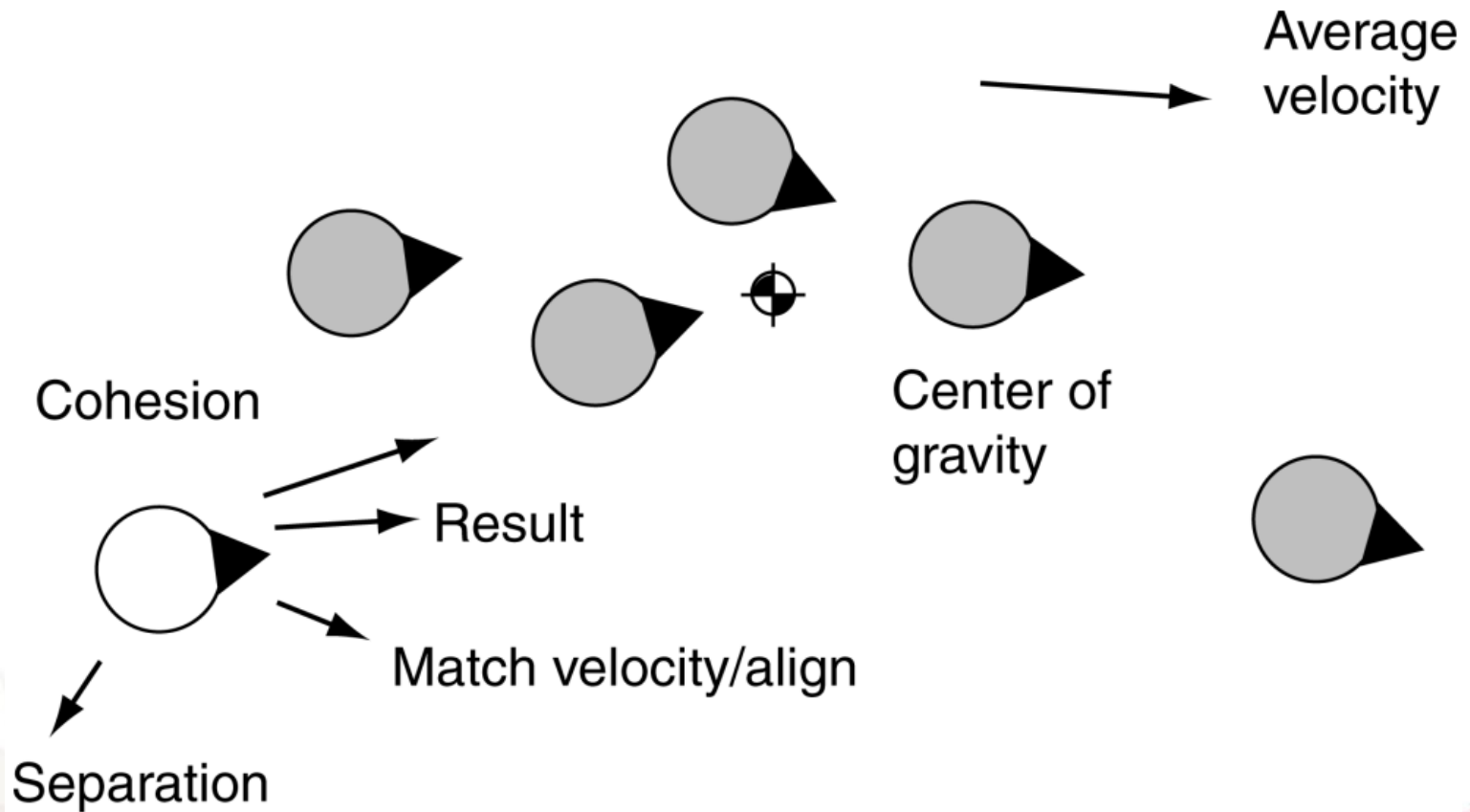
Flocking

- ❑ Original research by **Craig Reynolds**, to model movement patterns of flocks of simulated birds (“boids”).
- ❑ Flocking relies on simple weighted blend of 3 behaviours:
 - **Separation** – move away from boids that are too close
 - **Alignment** – move in the same direction and at the same velocity as flock
 - **Cohesion** – move towards the center of mass of the flock



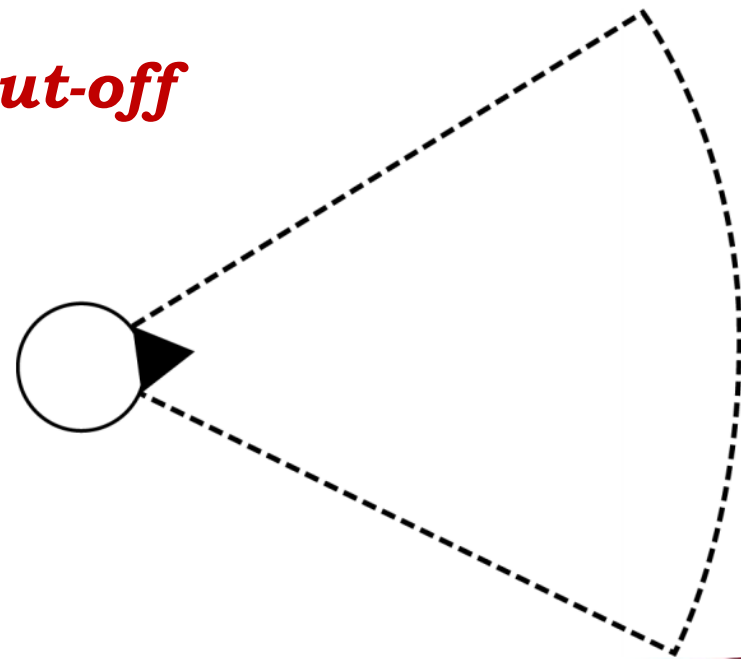
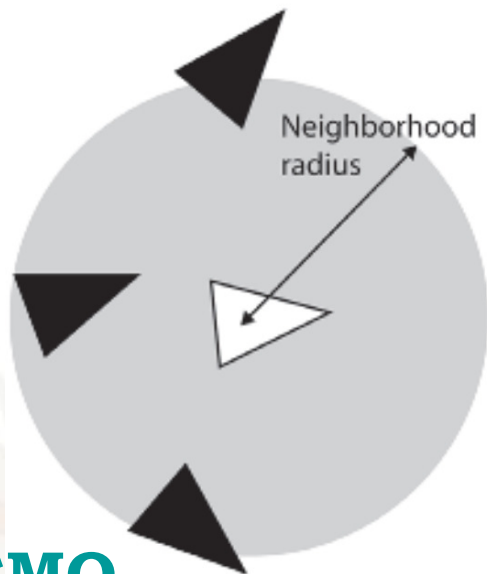
Flocking

- **Simple flocking:** *Equal weights for all*



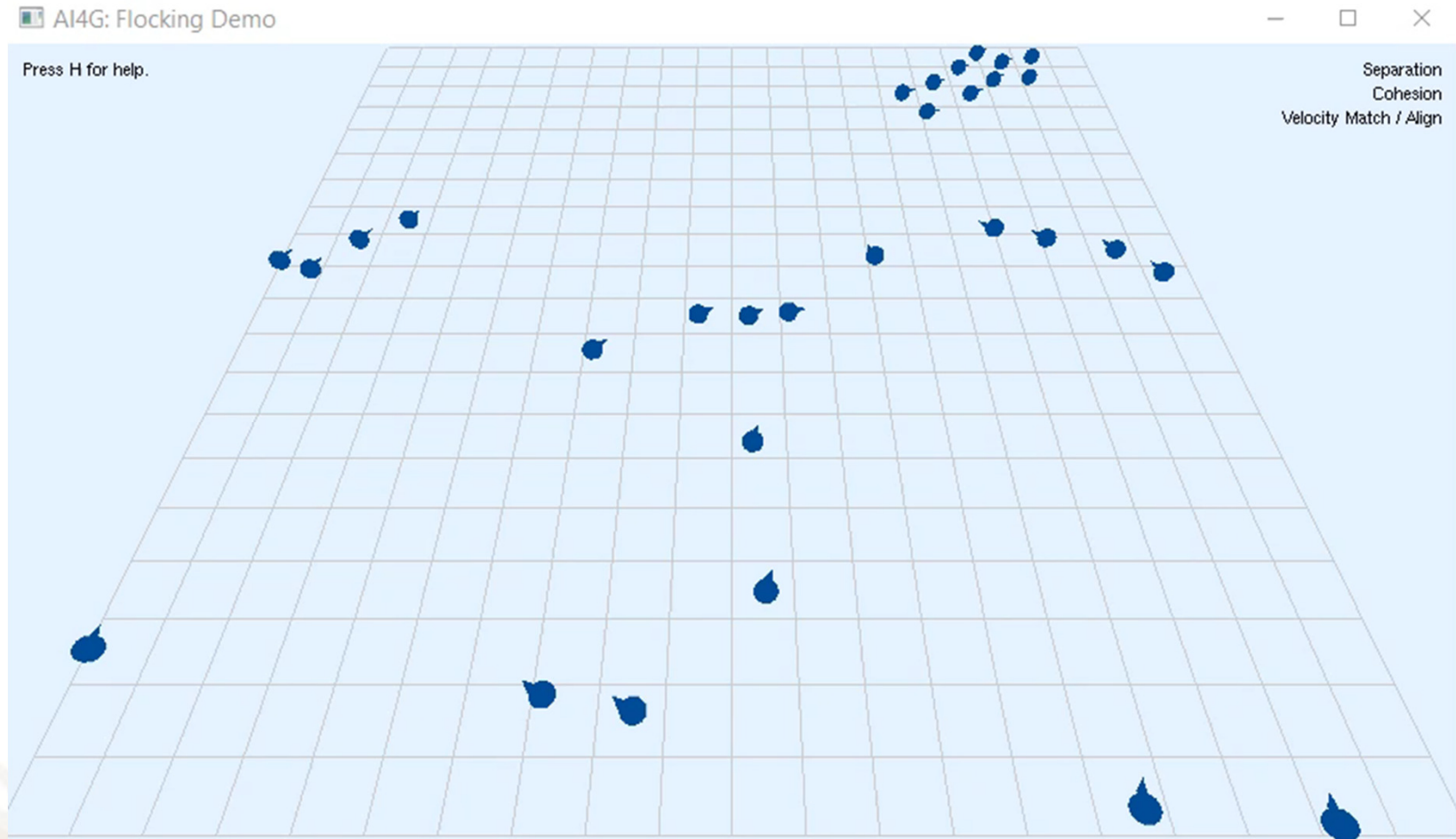
Flocking

- ❑ In most implementations, **flocking** behaviour is modified to ignore distant boids for efficiency
- ❑ A neighborhood is specified to consider only other boids within the area
- ❑ Shape: **Radius or angular cut-off**



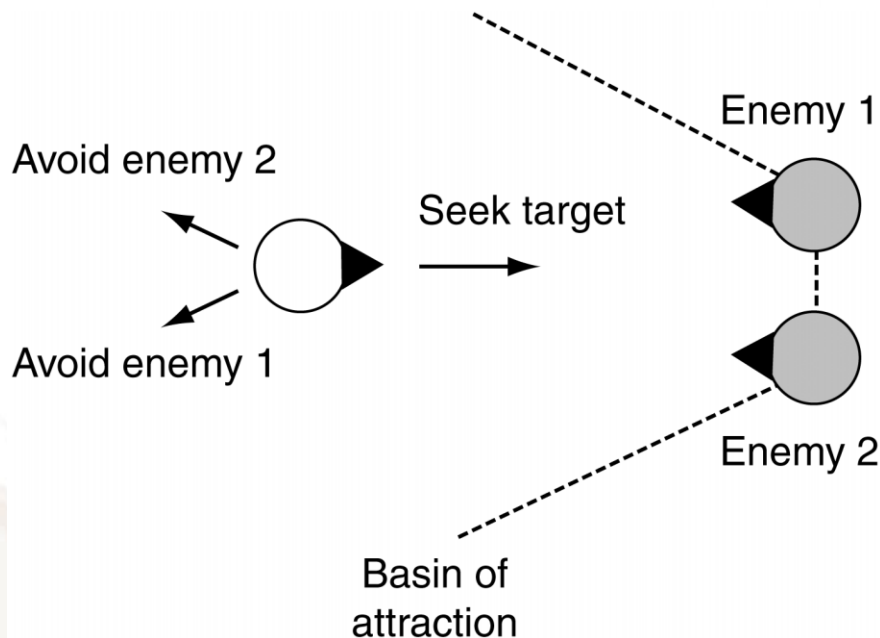
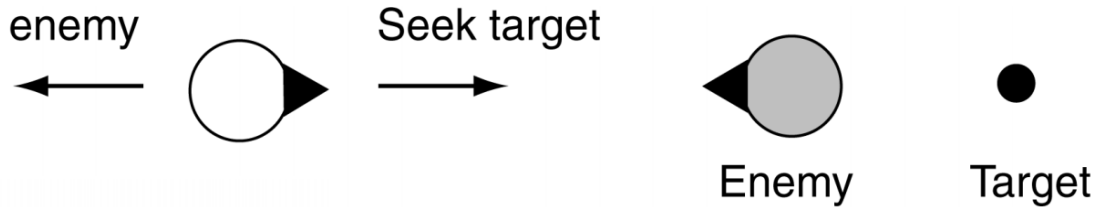
DEMO

Flocking



Equilibria Problems

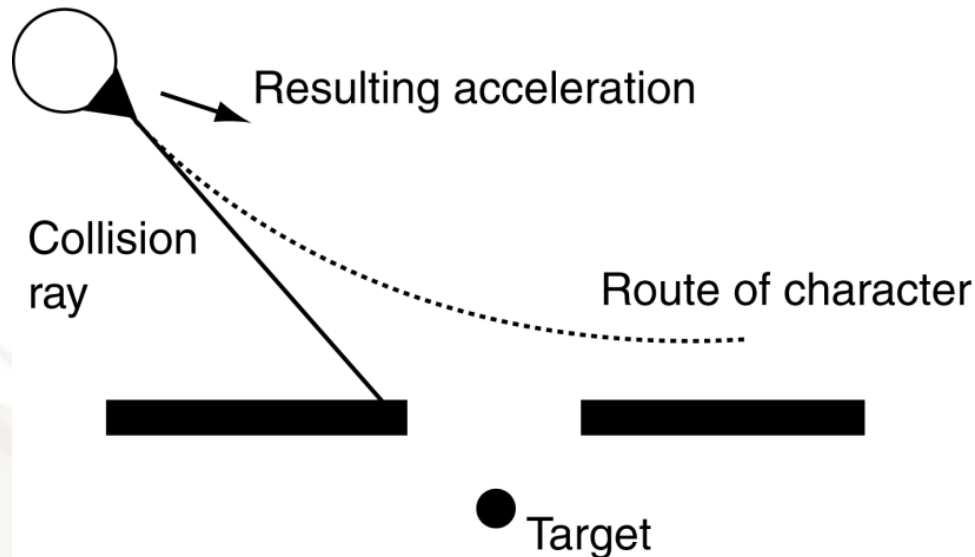
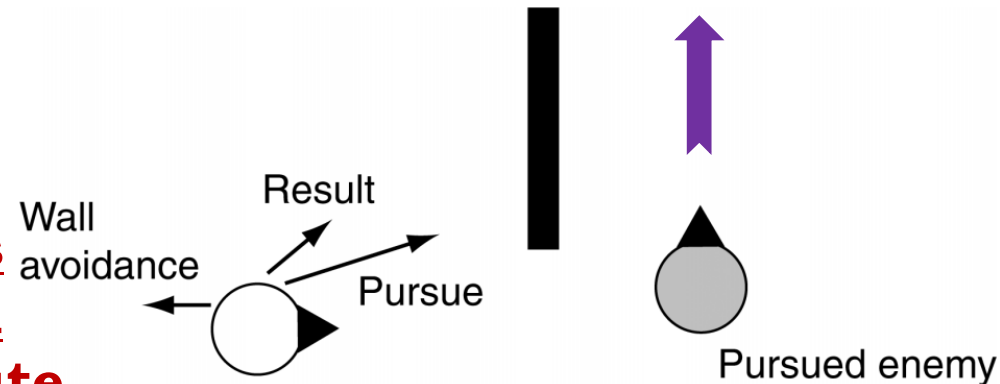
- **Unstable equilibria:** Character trying to do more than one thing at a time, resulting in doing nothing (as long as enemy is stationary) since the accelerations exactly cancel



- **Stable equilibria:** Character could make it out of equilibrium slightly, but heads back into equilibrium within basin of attraction

Constrained Envir...

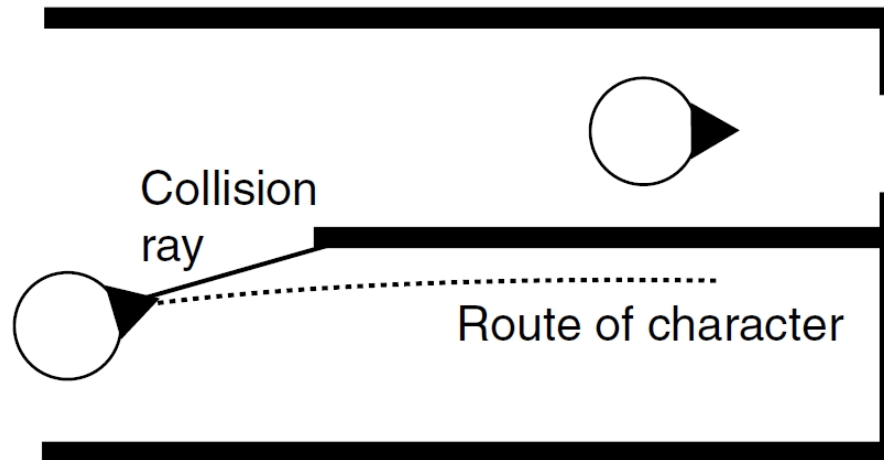
- ❑ **Obstacles Vs Target:**
Character tries to avoid obstacle while pursuing enemy. Blending causes resulting direction even farther from correct route to capture enemy



- ❑ **Narrow Doorways:**
Character tries to move at acute angles through narrow doorways to get to target. Obstacle avoidance causes the character to move past the door missing the target

Nearsightedness

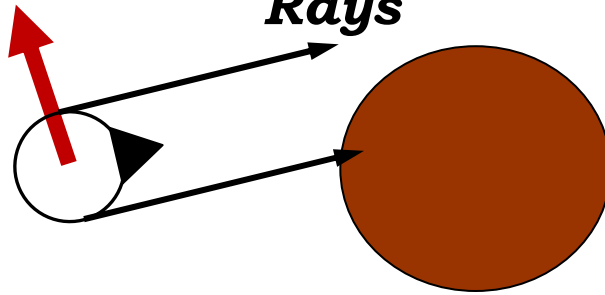
- ❑ **Nearsightedness:** Due to the behaviours acting locally in their immediate surroundings, **a character can avoid a wall**, but **takes the wrong side of the wall** due to method of computing change of orientation.
- ❑ *Does not realize the wrong path!*
- ❑ Can be addressed by *incorporating pathfinding*.



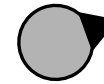
Judder

□ **Definition:** To vibrate with intensity

*Obstacle
Avoidance Collision
Rays*



Target



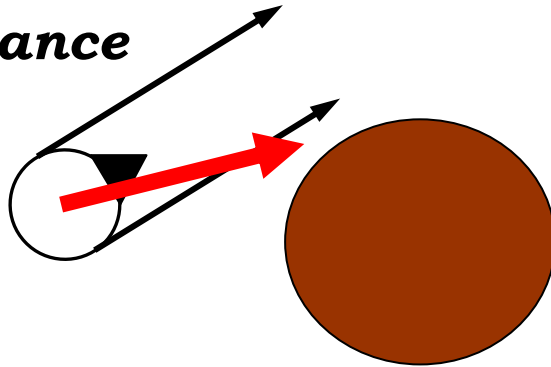
$t = 1$

Judder

□ **Definition:** To vibrate with intensity

*No
Obstacle
Avoidance*

*Collision
Rays*

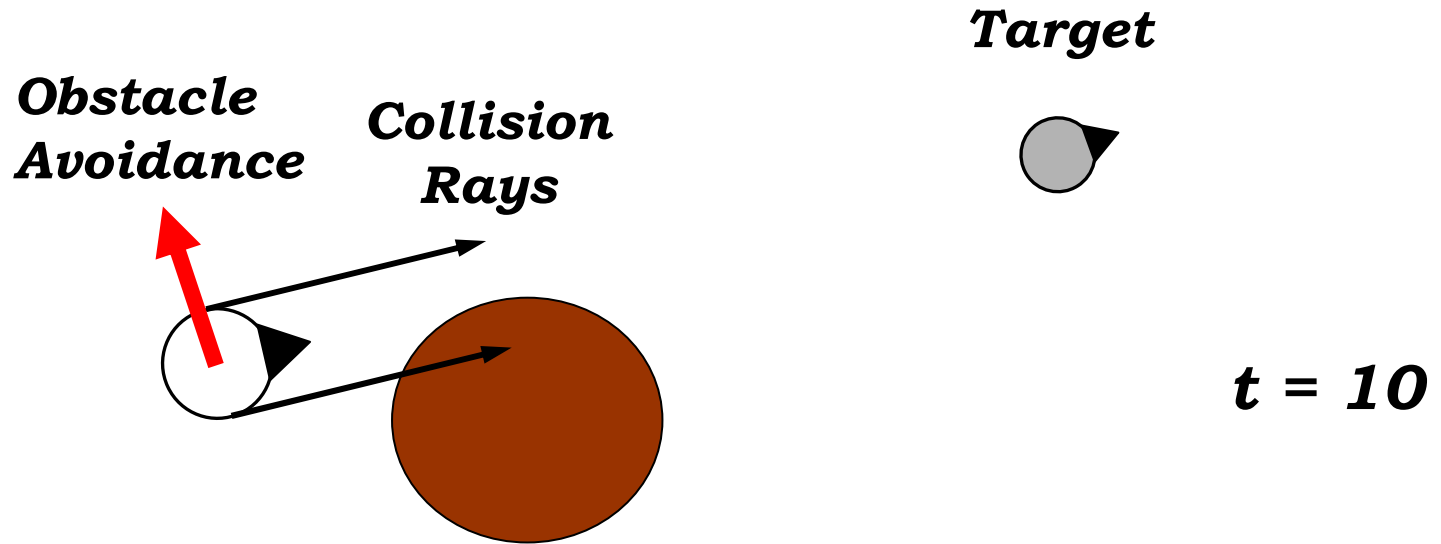


Target

$t = 5$

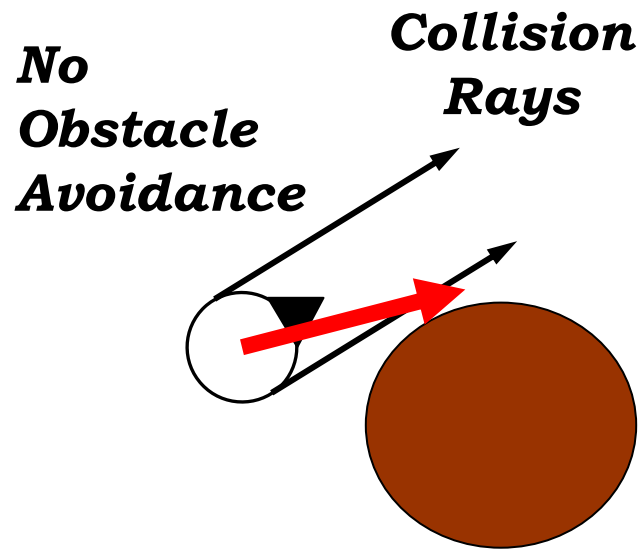
Judder

□ **Definition:** To vibrate with intensity



Judder

□ **Definition:** To vibrate with intensity



Target

$t = 15$

Priority-based Blending

- ❑ Some steering behaviours do not produce any acceleration as output in most situations (*Collision avoidance, Separation, etc.*) HOW?
- ❑ Others such as Seek and Evade which **always produced an acceleration**.
- ❑ So, when blended together some behaviour acceleration requests are *diluted by other behaviours*
- ❑ Example: **Seek** (always max acceleration) + **Collision Avoidance** (minimal change of movement to avoid).
- ❑ *Seek always dominates if blended equally!*

Priority-based Blending

❑ Idea:

- Arrange behaviours in groups with regular blending weights
- Place groups in order of priority, and consider each group accordingly
 1. If total result is very small (less than some threshold), ignore it and consider next group
 2. If total result is reasonable (more than some threshold), use the result to steer character

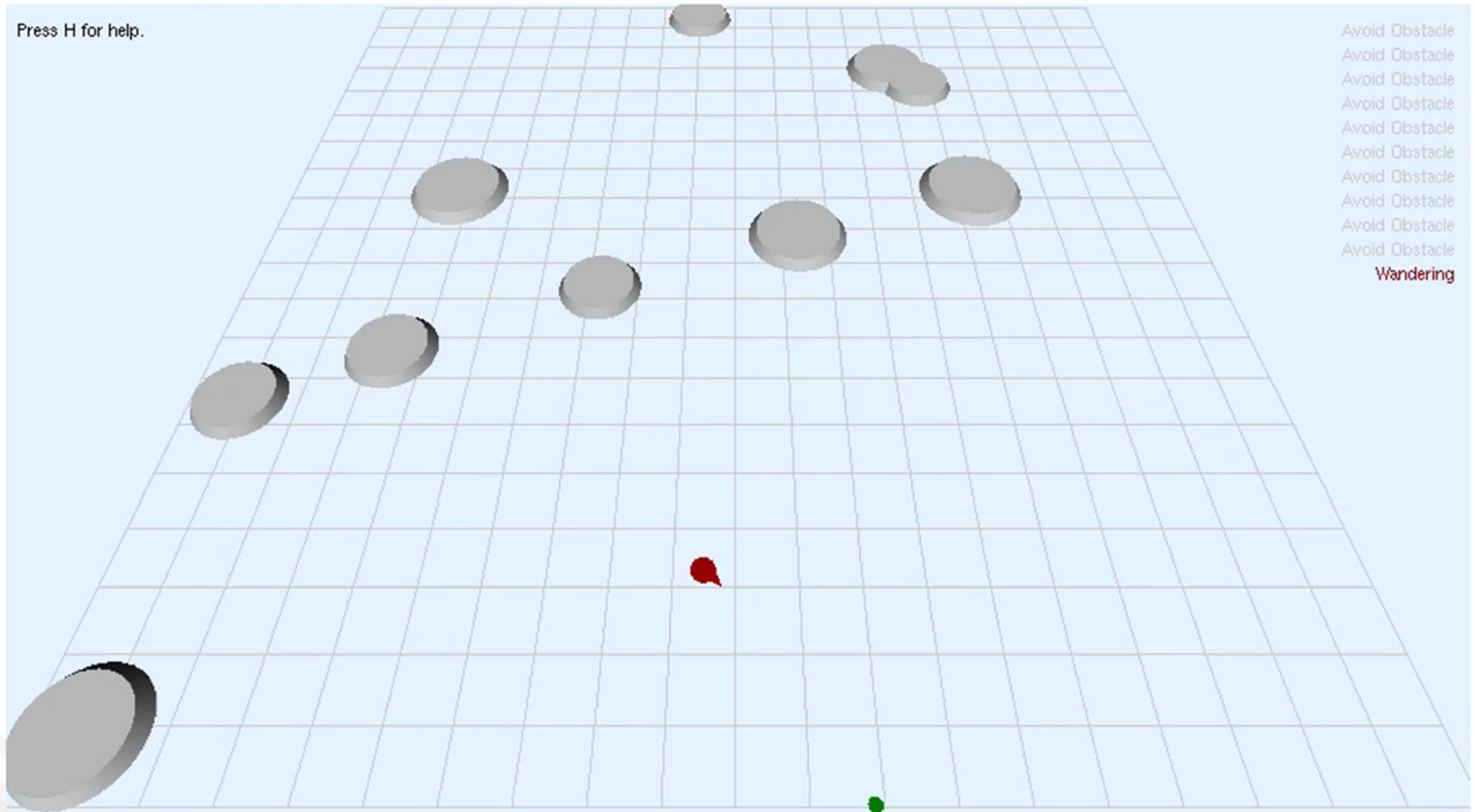
❑ Example: Pursuing character with 3 groups in priority → 1st: Collision avoidance; 2nd: Separation; 3rd: Pursuit

❑ DEMO

Priority-based Blending

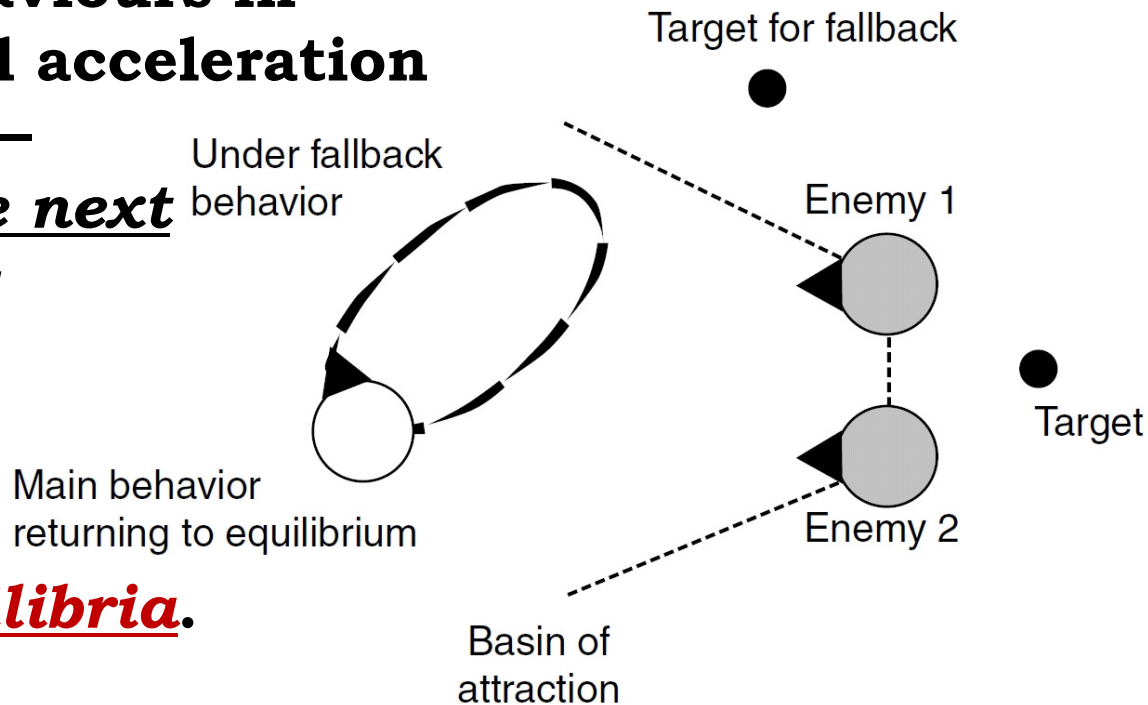
AI4G: Steering Priority Demo

Press H for help.



Equilibria Fallback

- Priority-based approach can cope with unstable equilibria problem.
- If a group of behaviours in equilibrium, total acceleration will be near zero – drop down to the next group in priority
- Example: Falling back to Wander
- But can't avoid large stable equilibria.



Variable Priorities

- ❑ If we want more control to be able **to avoid equilibria problems**, etc., instead of a fixed order to represent priority values, we can allow each group of behaviours return a dynamic priority value.
- ❑ These groups are then sorted by priority values and the algorithm continues as before.
- ❑ Although an obvious extension, the resultant practical advantages are minor and a full cooperative arbitration system (discussed next) may as well be used...

Cooperative Arbitration

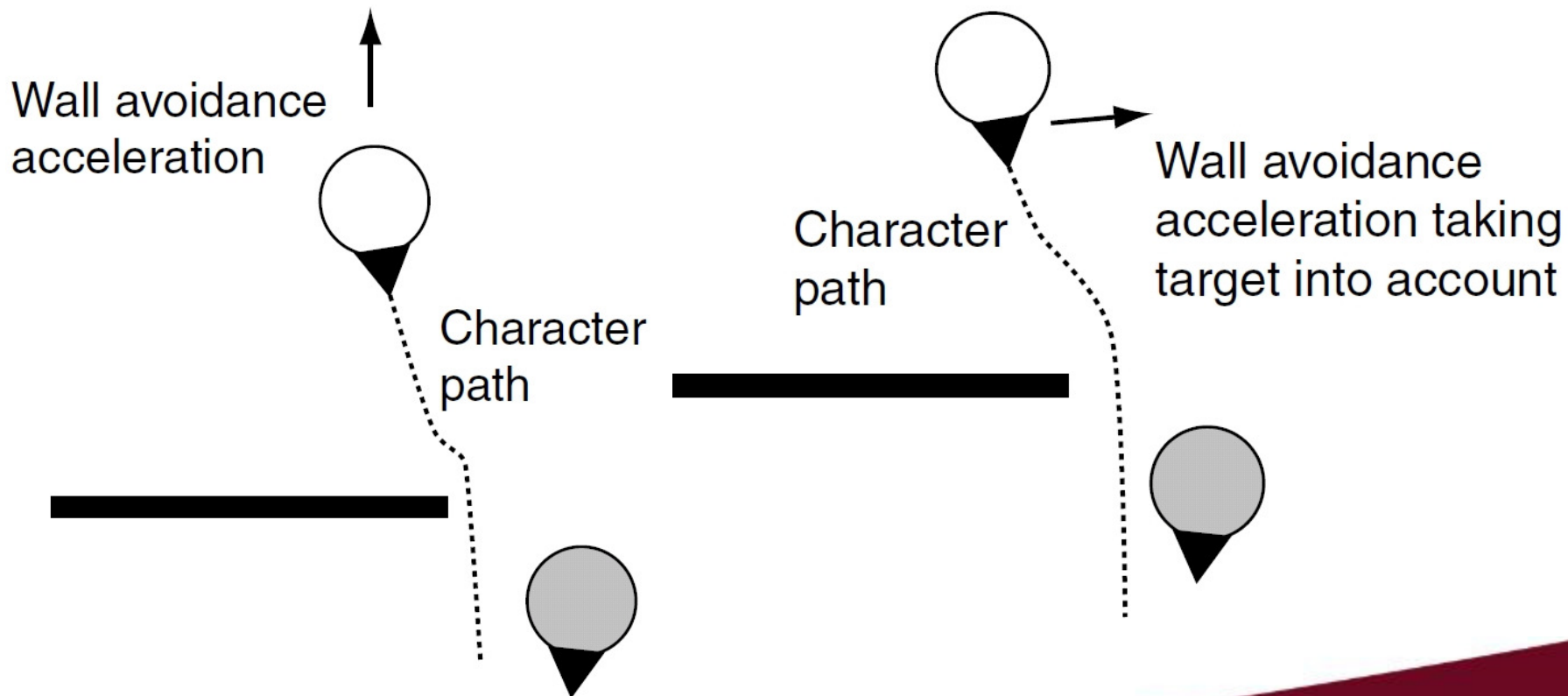
❑ **Main problems** with approaches so far:

- In weighted blending, one of the main behaviours *may be diluted* by the output of another behaviour
- In priority blending, a prioritized behaviour *may have a drastic effect* on the character movement (not smooth) when it changes to other behaviours of less priority

❑ Context-sensitivity or cooperation between different behaviours can help create more realistic and less-dramatic movement

Cooperative Arbitration

- Consider an example of a character chasing a target using Pursue while *avoiding collisions* with walls.



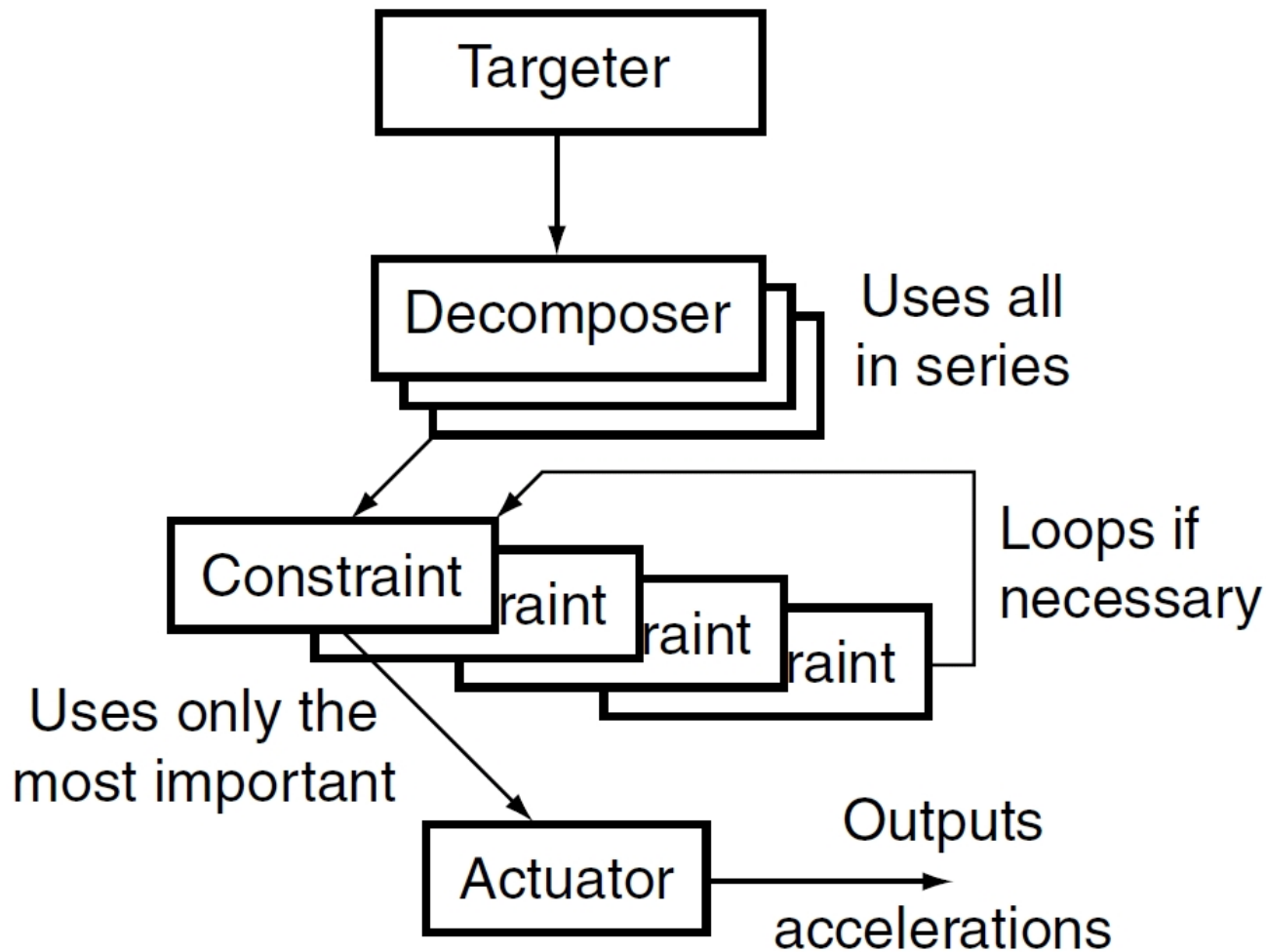
Cooperative Arbitration

- ❑ A cooperative arbitration approach that allows constructive interaction between steering behaviours.
- ❑ It *provides excellent performance* in a range of situations that are normally problematic, including *tight passages* and *integrating steering with pathfinding*.
- ❑ One of many possible approaches; not the only way.
- ❑ Not widely used, though.

Cooperative Arbitration

- ❑ ***Increases the complexity*** of the steering algorithm as the **simple building blocks must now collaborate**.
- ❑ **Collaborative arbitration implementations:**
 - **Decision making, Decision trees**
 - **State machines**
 - **Blackboard architectures: each behaviour is an expert that can read (from the blackboard) what other behaviours would like to do before having its own say**
- ❑ ***No de facto standard for games***
- ❑ For an example, we'll look at the **steering pipeline algorithm...**

Steering Pipeline



Steering Pipeline

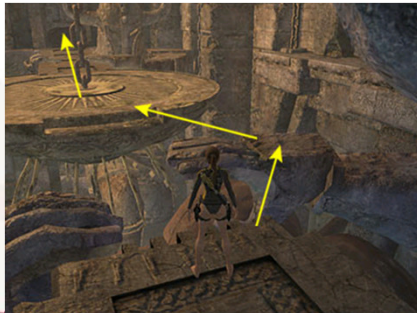
- ❑ There are four stages in the pipeline:
 1. the targeter works out where the top-level movement goal is,
 2. the decomposers provide sub-goals that lead to the main goal,
 3. the constraints limit the way a character can achieve a goal, and
 4. the actuator limits the physical movement capabilities of a character.
- ❑ In all but the final stage, there can be one or more components. All are steering behaviours.
- ❑ DEMO

Steering Pipeline



Jumping

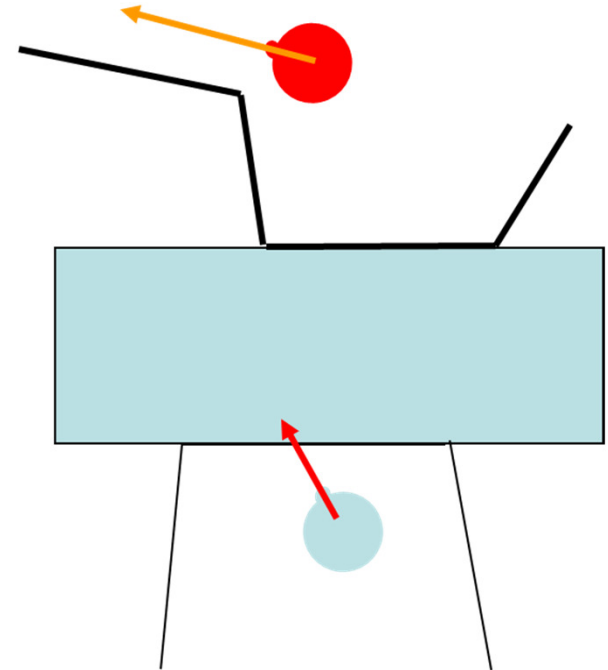
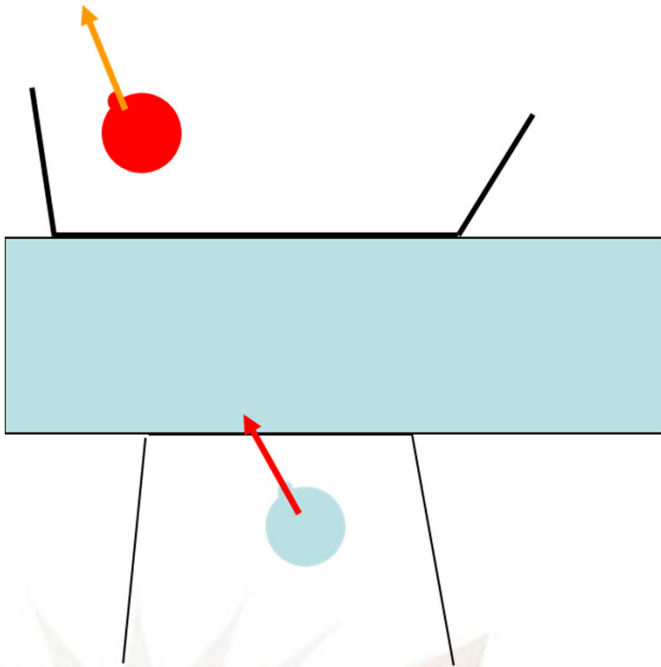
- ❑ First Person Shooter, Action, *etc.*, genre games need jumping as part of the character movement
- ❑ Jumping is not part of steering mechanisms
- ❑ Jumps *can be failures*
 - Character tries to jump from one platform to another but fails
 - *Cost of error larger* than for steering
 - Slight errors using steering behaviours while pursuing are corrected almost immediately



<http://www.tombraiderchronicles.com/underworld/walkthrough/level02-2.html>

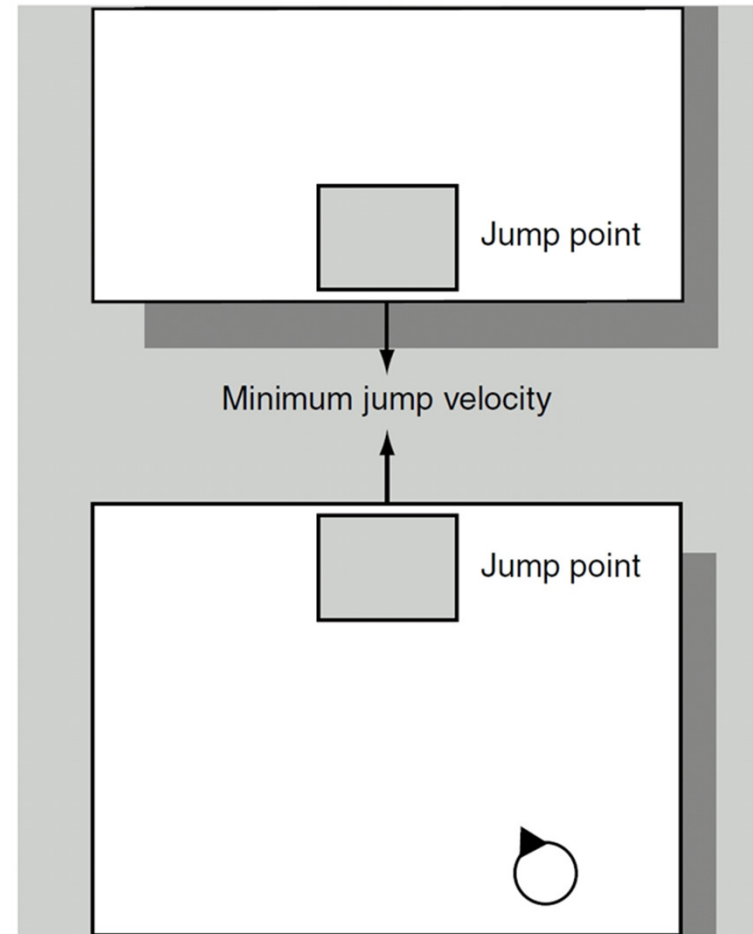
Jumping

- ❑ Character must set up for jump on the right, but not on the left:



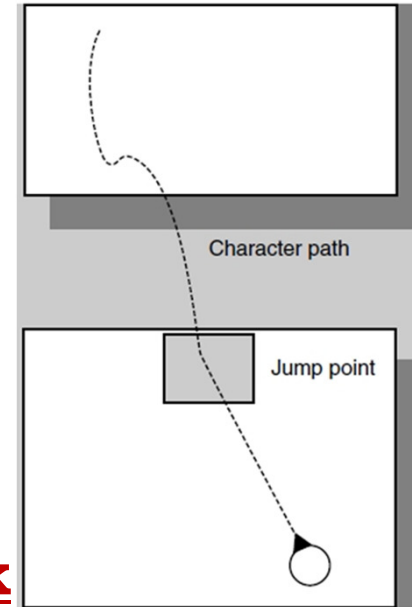
Jump Points

- ❑ The simplest support for jumps puts the onus on the level designer.
- ❑ Locations in the game level are labelled as being jump points. These regions need to be manually placed.
- ❑ If characters can move at many different speeds, then jump points also have an associated minimum velocity set.



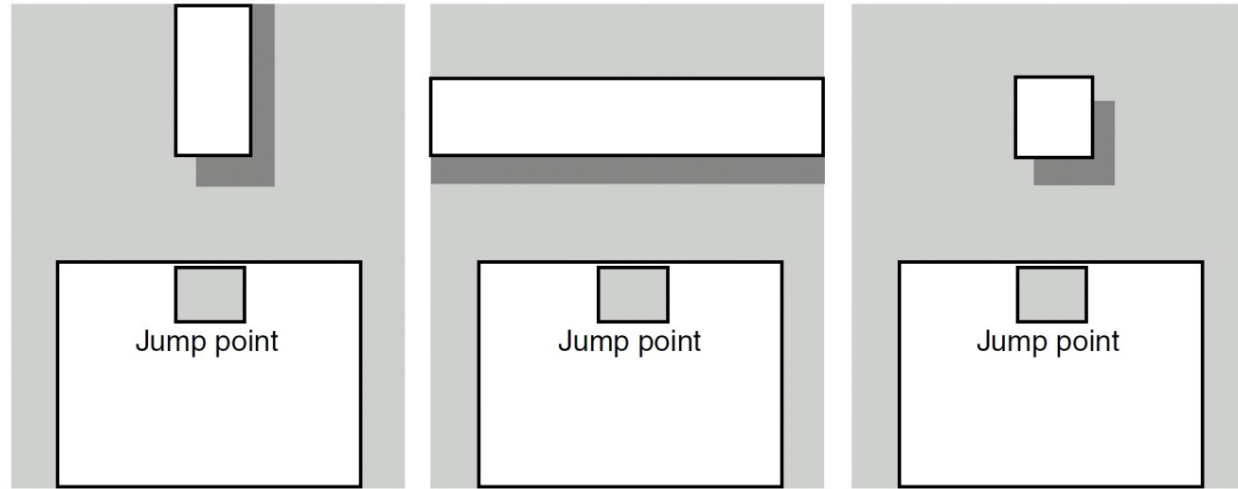
Jumping

- ❑ Character uses the *velocity matching* steering mechanism to make the jump:
 - Character decides to take the jump
 - *Pathfinding* decides that character needs to jump (this gives also *type of jump*), OR
 - Simple steering mechanism *drives character over edge* (this needs look ahead to determine type of jump)
- ❑ New steering behavior to do velocity matching to reach the jump point with the correct velocity
- ❑ When character reaches jump point, *a jump action is executed*



Jumping - Weakness

- ❑ Jumps are *difficult to make* if:



- ❑ Create jump points for characters in level design
 - Player characters can try to make difficult jumps, but *AI characters cannot*
 - Minimize number of occasions where this limitation becomes obvious
 - Level design needs to *hide weaknesses in the AI*

Jumping – Landing Pads

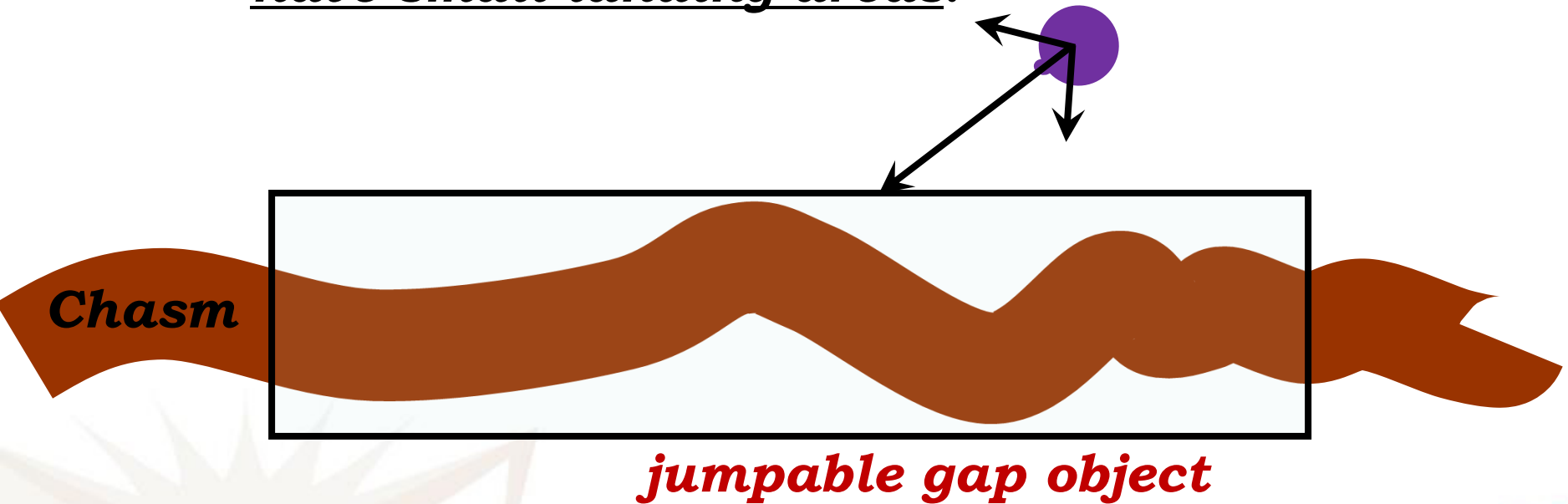
- ❑ A better alternative uses *pairs of jump points and landing pads* (very much like jump points)
- ❑ Rather than require the level designer to set up the required velocity, we can leave that up to the character.
 - When character decides to make the jump, add an additional step:
 - Use trajectory prediction code to calculate velocity required to reach landing area
 - This allows characters to take their own physics (weight, load, strength) into account
- ❑ Use velocity matching steering algorithm

Jumping – Hole Fillers

- ❑ Another approach allows characters to *choose their own jump points*
- ❑ Create an invisible “*jumpable gap*” object to fill the hole as part of certain obstacles
- ❑ Change obstacle avoidance behaviour to a “*jump detector*” behaviour:
 - When detecting collision with a jumpable gap, character runs towards gap at full speed.
 - Just before the gap, character leaps into air
- ❑ Characters are not limited to a particular set of locations from which they can jump.

Jumping – Hole Fillers

- ❑ Works well if landing area is large
- ❑ Fails for small landing areas
 - In this case, ensure that level design does not have small landing areas.



Coordinated Movement

- ❑ Done by groups of characters
- ❑ Coordinated movement can occur at two levels
 - can result from **individual decisions** *that complement each other* (so that their movement looks coordinated)
 - can result from decisions made by the group as a whole
- ❑ We will focus on formation motion which is the movement of a group of characters so that they retain some group organization.
- ❑ We will consider tactical decision making later.

Fixed Formations

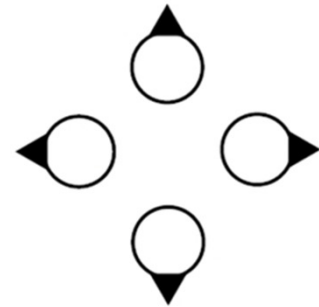
□ Simplest kind: Fixed geometric formations

➤ Usually with *a designated leader*

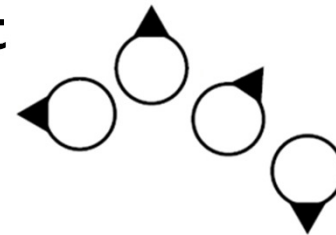
➤ **Leader** moves as **Line**
an individual,
everybody else
moves based on leader's position



**Defensive
Circle**



➤ The movement of the leader character should take into account the fact that it is carrying the other characters with it.

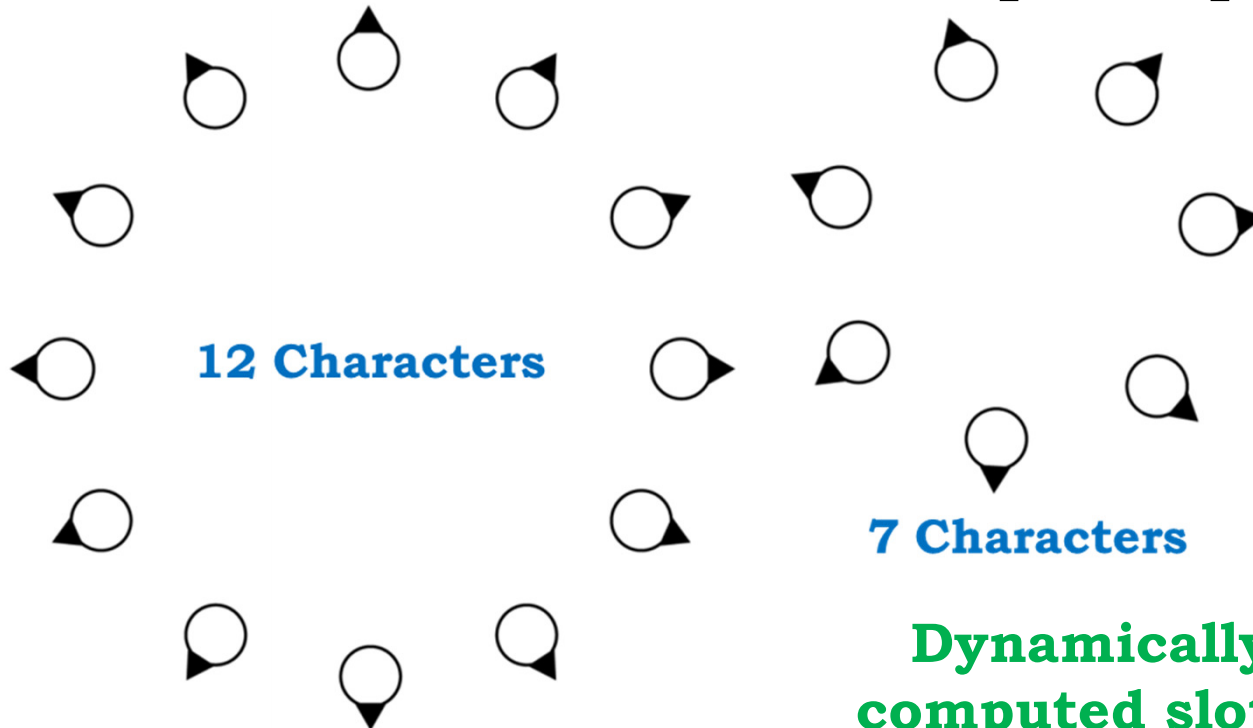


➤ Actual positions determined from formation **V or "Finger Four"**

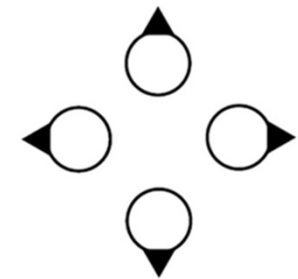
**Two abreast
in cover**

Scalable Formations

- The exact structure of a formation will depend on the number of characters participating in it.



12 Characters



4 Characters

7 Characters

Dynamically
computed slot
locations

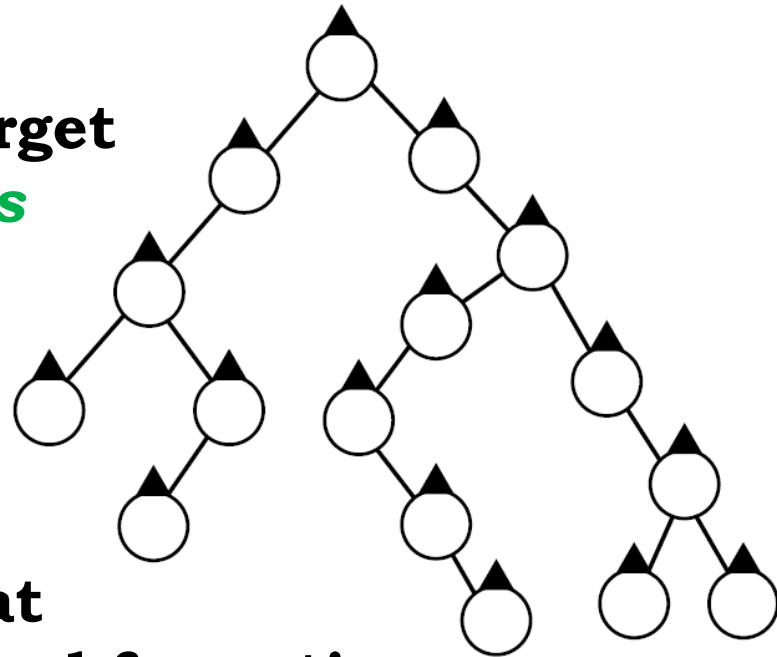
With 100 defenders, it may be *possible to structure the formation in several concentric rings*



<http://www.moddb.com/mods/rearm/videos>

Emergent Formations

- ❑ Emergent Formations (different solution to scalability)
 - Each character has its own steering system
 - Arrive Behavior
 - Characters select their target *based on other characters in group*
 - Allows characters to avoid obstacles individually
 - *Difficult to get rules* that do not lead to pathological formations
 - May not have a leader

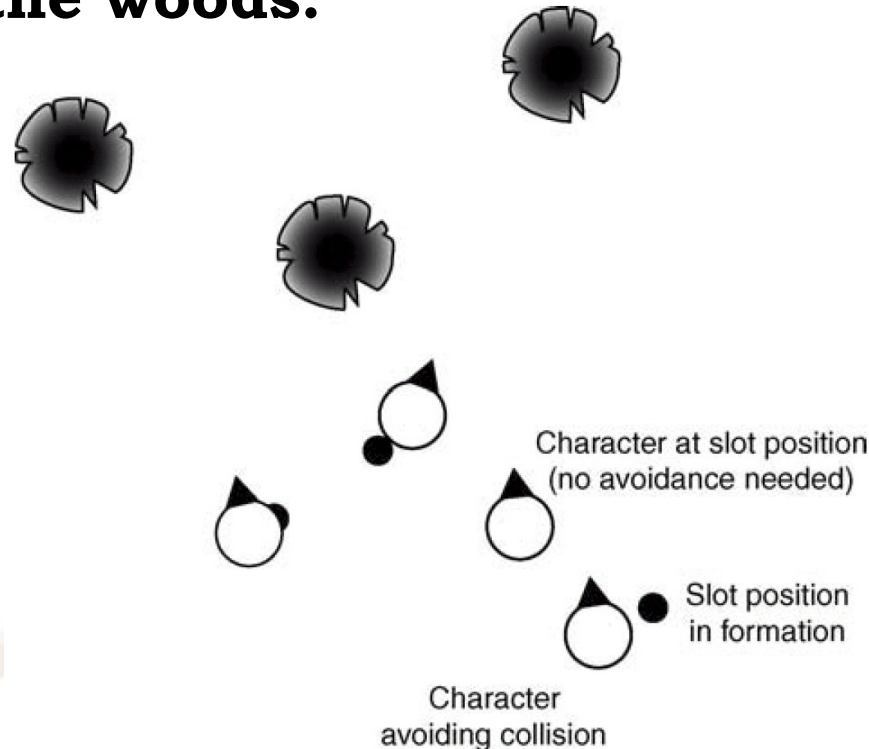


Two-Level Formation Steering

- ❑ Combines strict geometric formations with the flexibility of an emergent approach
- ❑ Use a strict formation, as before
- ❑ Initially, have a character that serves as **formation leader** (formation moves and turns with leader)
- ❑ Individual characters use emergent approach to steer individually
 - Use **arrival**, **avoidance**, and **separation** behavior to reach target slot (based on strict formation)
- ❑ Has **difficulties** when (**non-leader**) characters **run into obstacles** and cannot keep up with group

Two-Level Formation Steering

- E.g., a number of NPCs moving in V formation through the woods.



Two-Level Formation Steering

- ❑ In the example above, *if the leader needs to move sideways* to avoid a tree, then all the slots/characters in the formation will also lurch sideways
- ❑ Instead: use a *pseudo-character with a fixed location*. Its position will be the pattern's anchor point - used to lay out the formation pattern, slot locations, and formation rotation.
- ❑ A separate steering system will move the anchor point which is impervious to obstacles, etc., but aware of large obstacles such as walls.
- ❑ All characters then react in the same way to their slots.

Two-Level Formation Steering

- ❑ Thus far, information has only flowed from the formation to the characters.
- ❑ If some characters are having problems keeping up, say, manoeuvring around the small obstacles ignored by the anchor point, the formation will be oblivious.
- ❑ To prevent problems for characters keeping up:
 - A. Slow the formation down (about half of character speed), or
 - B. Moderate movement of formation based on current positions of characters in slot

Two-Level Formation Steering

- ❑ Latter approach: reset kinematics of anchor point
 - Base the position, orientation, velocity of anchor points on the **average position**, p_c , and **average velocity**, v_c , of the characters in the slots
- ❑ Choosing exactly the average means that characters are almost in position, so that they move slowly towards their slot position.
- ❑ Anchor point moves even slower, etc.
- ❑ Move anchor point ahead of the average position for moving formations:

$$p_{\text{anchor}} = p_c + k_{\text{offset}} v_c$$

Two-Level Formation Steering

- Each character is assigned a slot coordinate p_{si} that will be its target position for its individual Arrive behaviour.

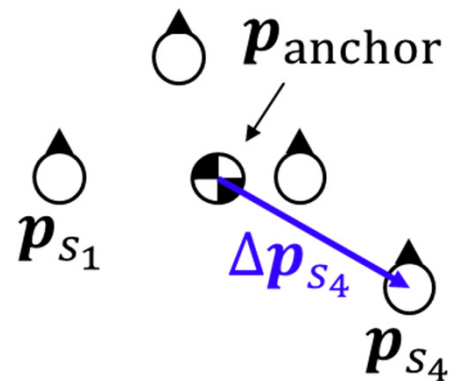
- Each slot coordinate *w.r.t.* the anchor point (assuming default orientation) is

$$\Delta p_{si} = p_{si} - p_{\text{anchor}}$$

- This **does not change** unless **different combinations of slots are occupied**, or a **character may be killed off** (changing p_c)

- Then the slot coordinate is $p_{si} = p_{\text{anchor}} + \Delta p_{si}$

- This position is rotated around p_{anchor} by the **formation orientation**.



Two-Level Formation Steering

- If the formation is to stop, we could set anchor point to average position for stationary formations:

$$p_{\text{anchor}} = p_c$$

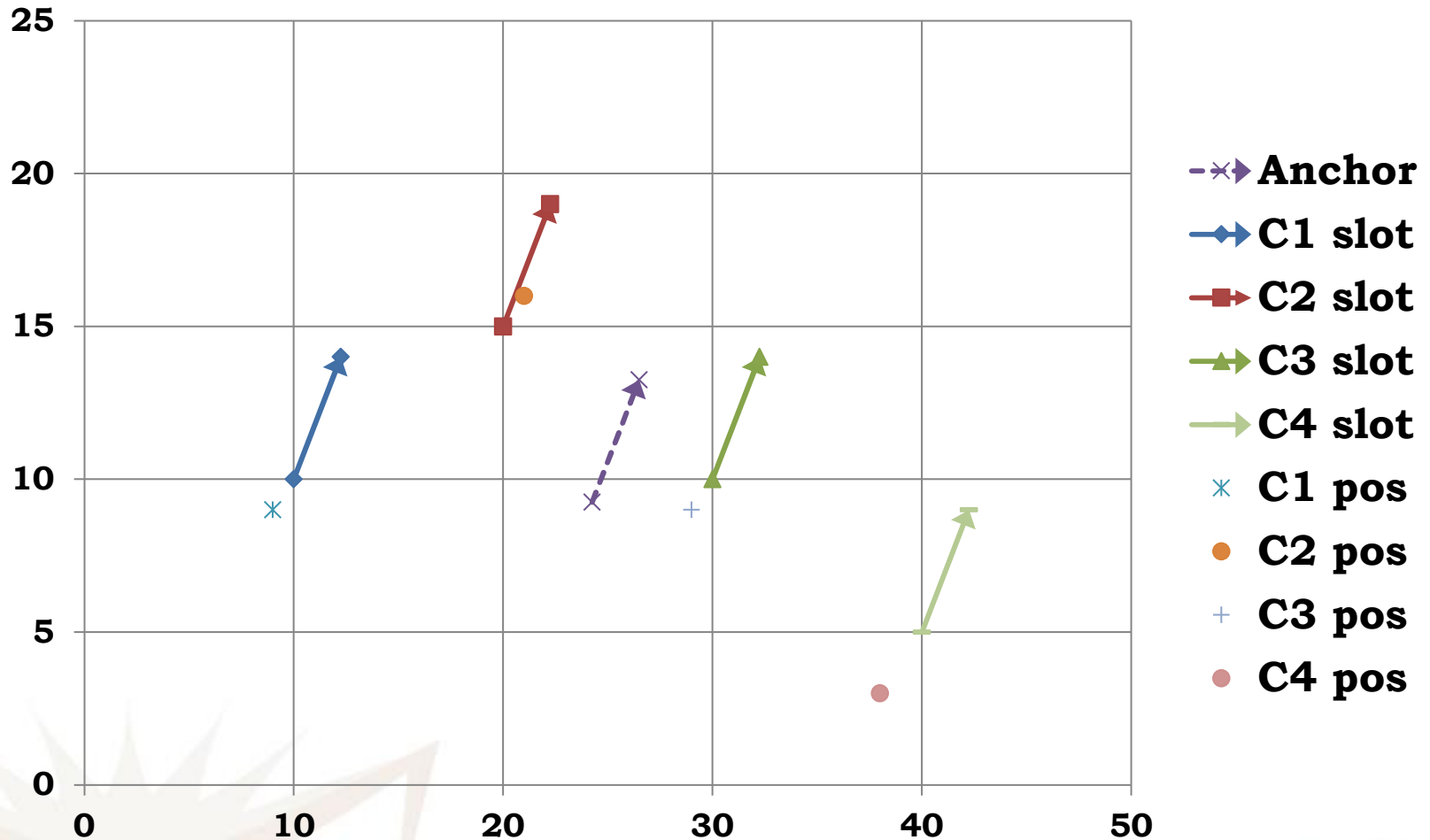
- The slots will then be updated which will then move the anchor point, causing the whole formation to drift.
- To correct this drift, the anchor point is set to the average position of the occupied slots (rather than the positions of the characters).
- Also, ensure that anchor point orientation is average orientation of slots, or formation will spin around.

Example

- Consider a V formation of four characters with a center of mass (or the current update's anchor position if no characters are lagging) of $p_c = (24.25, 9.25)$ and an average velocity of $v_c = (2.25, 4)$, so that the next update's anchor position (using $k_{\text{offset}} = 1$) is $p_{\text{anchor}} = (26.5, 13.25)$.
- Consider character 1 whose slot position relative to the center of mass is $\Delta p_{s_1} = (-14.25, 0.75)$. It's next slot position (used as the target position for that character's Arrive behaviour) is

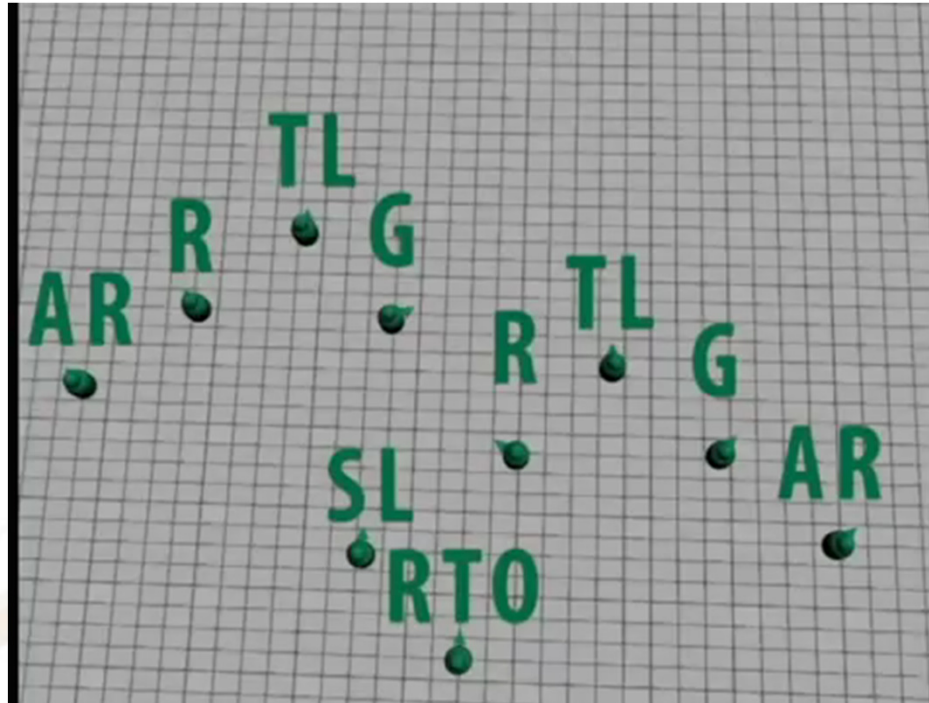
$$\begin{aligned} p_{\text{anchor}} + \Delta p_{s_1} &= (26.5, 13.25) + (-14.25, 0.75) \\ &= (12.24, 14) \end{aligned}$$

Example



Extending to More than Two Levels

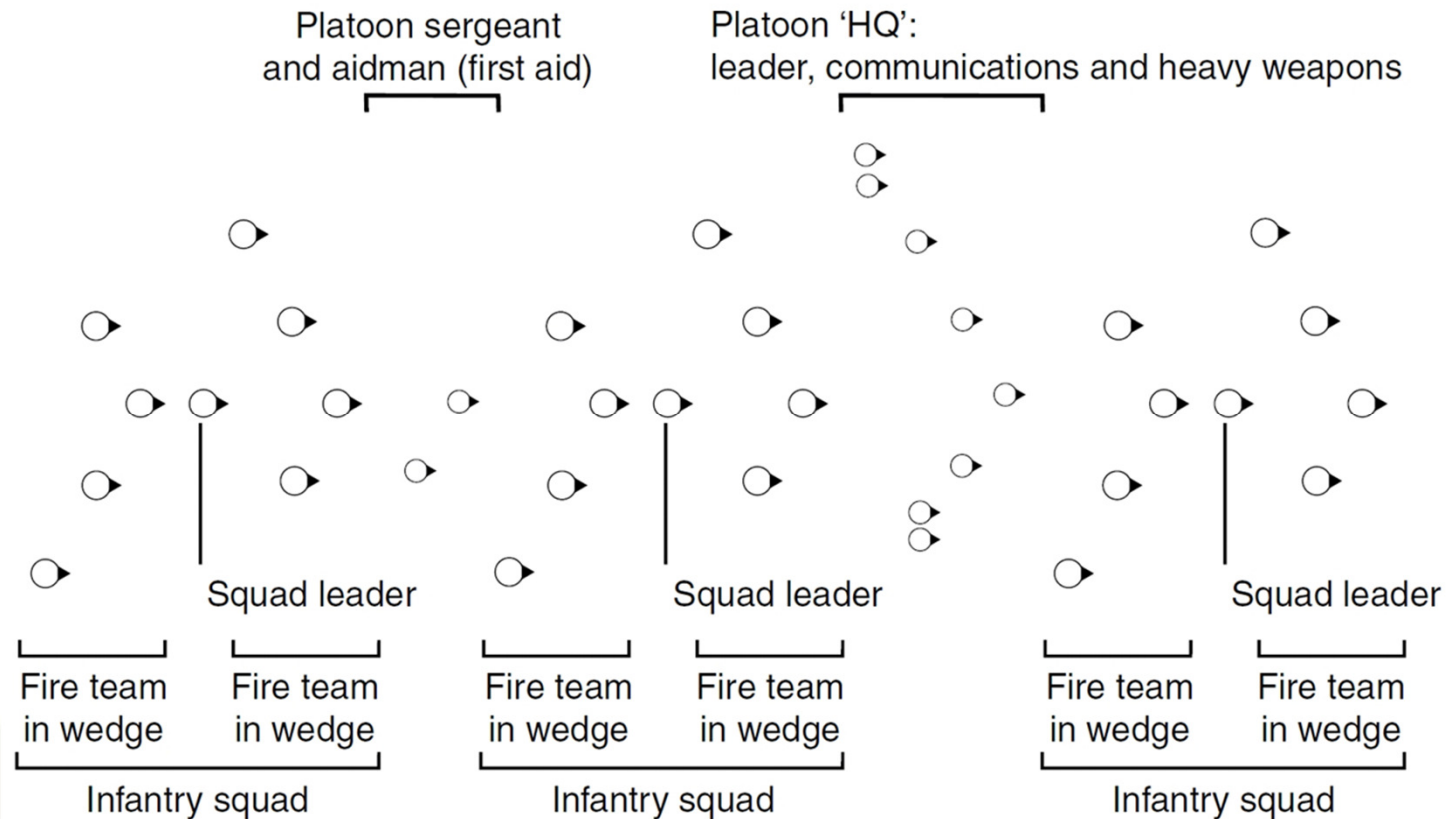
- ❑ **The two-level steering system can be extended to more levels**



Extending to More than Two Levels

- ❑ Needed for military simulation games with lots of units
 - Consult public military manuals how to organize a platoon in different squads
- ❑ Slot positions now distinguish between roles of characters:
 - E.g. **Fire teams**, **squad leader**, communication, **platoon leader**, heavy weapons team, ...

Nesting Formations to Greater Depth



Dynamic Formation Patterns

- ❑ Dynamic slots and playbooks
 - Not all games can use constant formations
 - Slots can be dynamic, moving relative to the anchor point of the formation
 - E.g. Fielders in a textbook baseball double play
 - E.g. Corner kick strategies in soccer
 - E.g. The basis of tactical movement
- ❑ Characters can “jump” to their new locations using **Arrive** behaviour to move there.
- ❑ An element of time needs to be introduced.

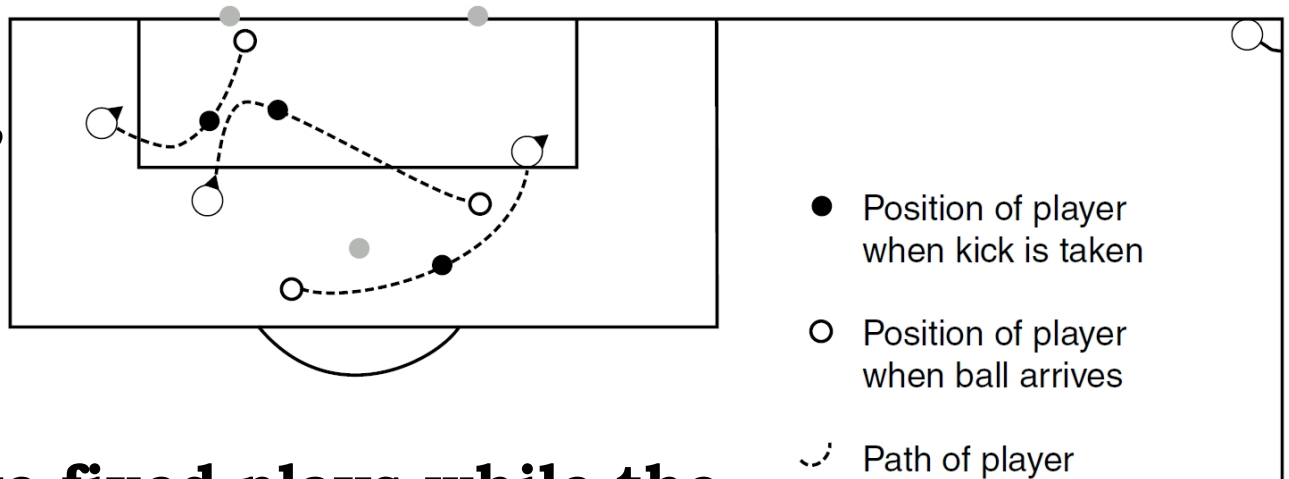
Dynamic Formation Patterns

- ❑ The moving slot positions need not be completely pre-defined. The slot movement itself can be determined dynamically by a controlling AI routine, moving the characters, say, to react to a tactical situation.

- ❑ In practice, **a mixture is used.**

**Below,
three**

**players have fixed plays while the
*others react to the defending
team's players.***

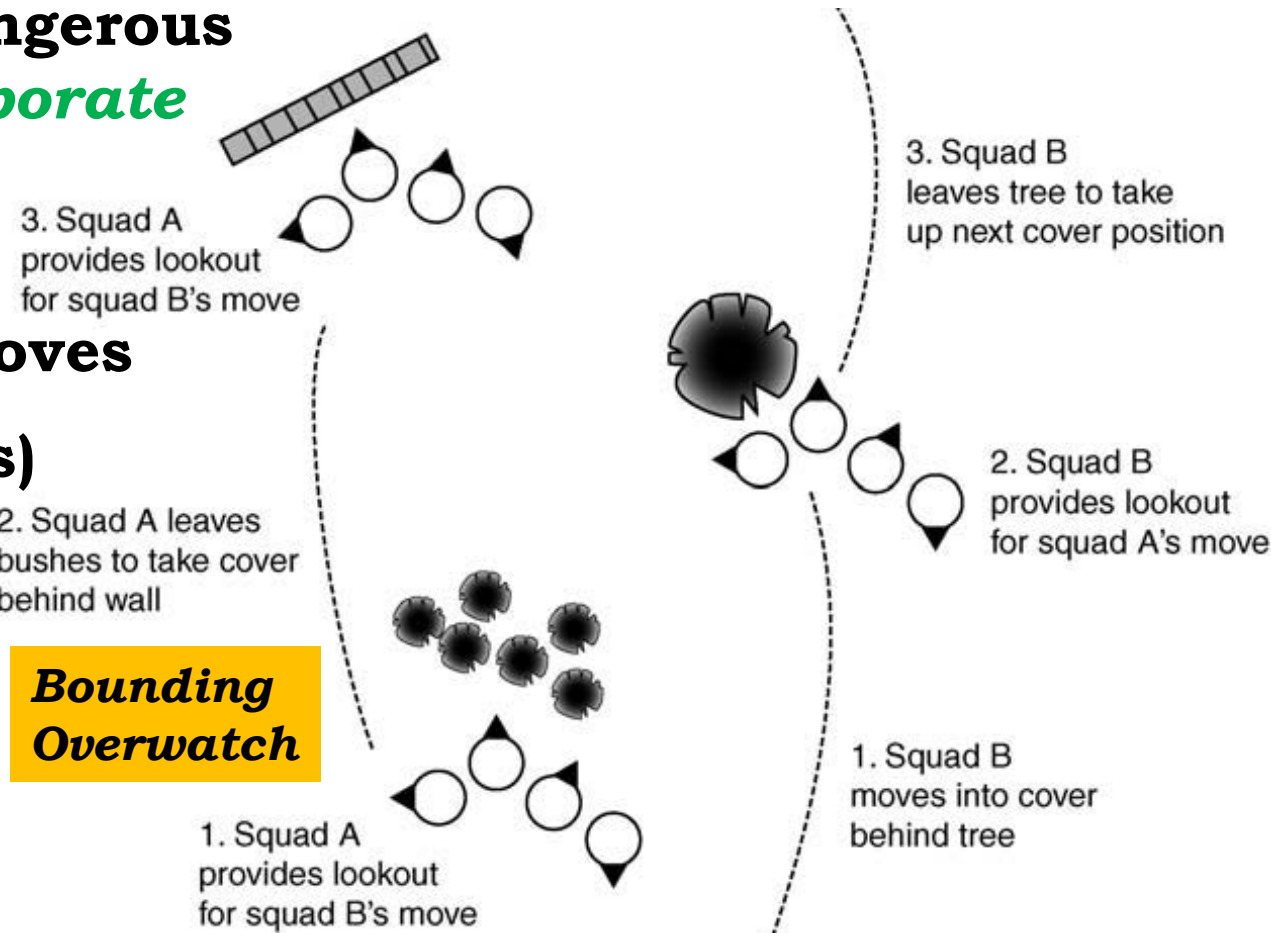


Tactical Movement

- ❑ Squads in dangerous terrain *collaborate with other squads*

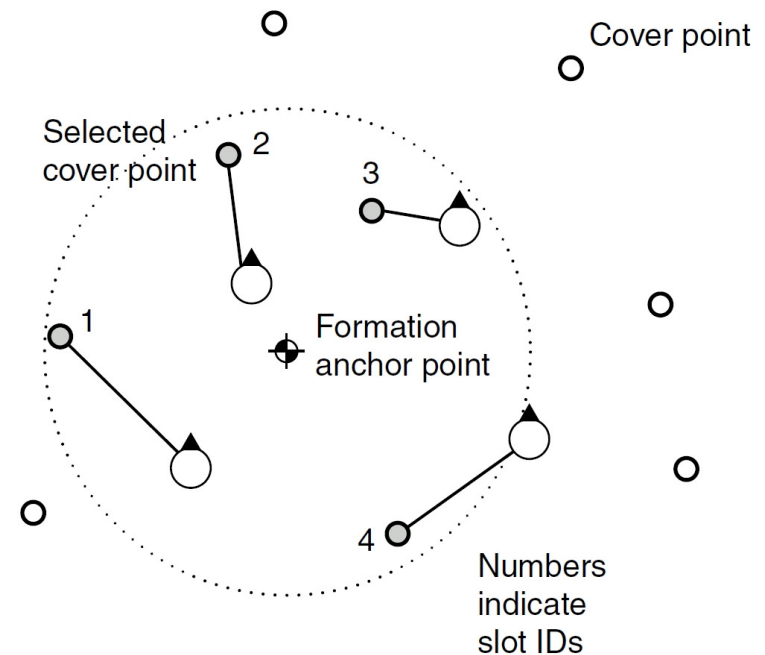
- ❑ One squad moves

- ❑ Other squad(s) provide *lookout and fire cover*



Squad-Based Movement

- ❑ Dynamic formation patterns can also be used to create a very simple but effective approximation of bounding overwatch. The formation slots move between cover points (where a character is safe).
- ❑ The closest set of cover points to the anchor point is used.
- ❑ The formation pattern uses of this set for the location of each slot.
- ❑ The formation is now linked to its environment



Squad-Based Movement

- ❑ As the formation moves, the set of cover points will change. When one cover point leaves, the slot assigned to it will now be assigned to the newly arriving cover point.
- ❑ The anchor point should move slowly compared to *the individual characters*
- ❑ Alternative anchor point: lead character.

