AI in First-Person Shooter Games

Based in part on material developed by
John McCloskey
Jeffrey Miller
Amish Prasad
& Lars Linden

FPS AI Architecture

- Animation
- Movement
- Combat
- Behavior

Animation Layer

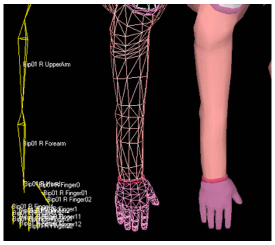
- Controls the player’s body
- Must resolve conflicts between animations
  - Dynamic blending
- Actions often need to be specialized
  - Parameterize existing animations
  - Taking control of specific body parts
  - Handling inverse kinematics

AI Components: Animation

- NPC models built by artists
  - Use tools such as “3D Studio Max” or “Maya”
- Models are are constructed from bones
- Bones are connected by articulated joints.
  - The movement of the joints is constrained by their interconnectivity.
  - See George Bush ragdoll physics demo.
- The skeletal system is covered by a mesh of textured polygons (“skeletal animation”.) *Half-Life* was one of the first games to demonstrate this.

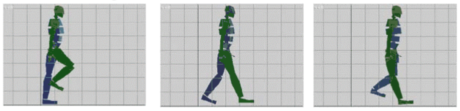
AI Components: Animation

- Example:



AI Components: Animation

- Animation sequences are generated by defining how joints should articulate through time
- Walking sequence:



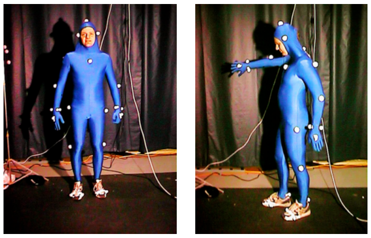
AI Components: Animation

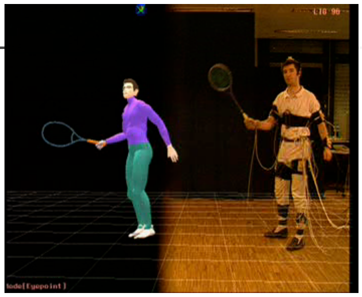
Animation sequences for a model are either:

- Hand generated by a computer animator
- Recorded from real human (or animal) movements and applied to a skeletal system ("motion capture")

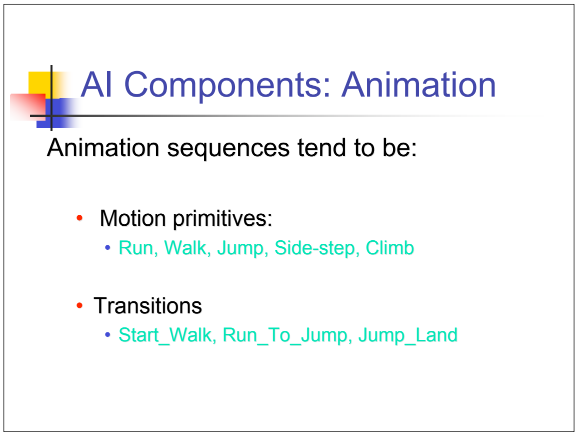
AI Components: Animation

- Motion Capture:





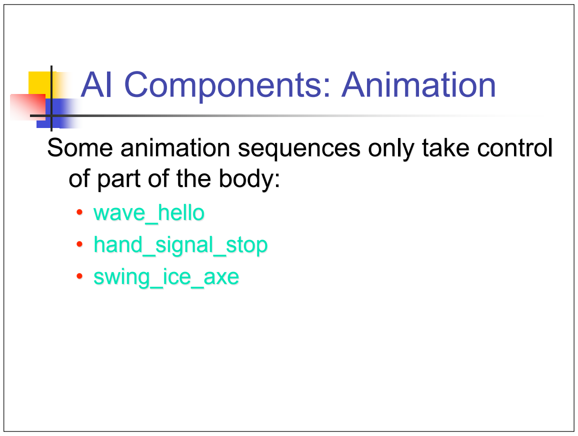
Tom Molet (EGCAS '96)



AI Components: Animation

Animation sequences tend to be:

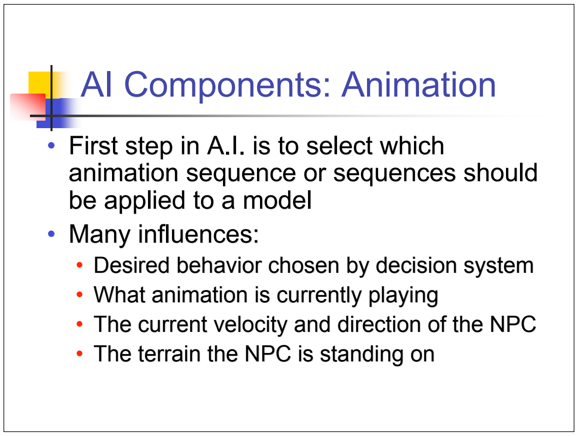
- Motion primitives:
  - Run, Walk, Jump, Side-step, Climb
- Transitions
  - Start_Walk, Run_To_Jump, Jump_Land



AI Components: Animation

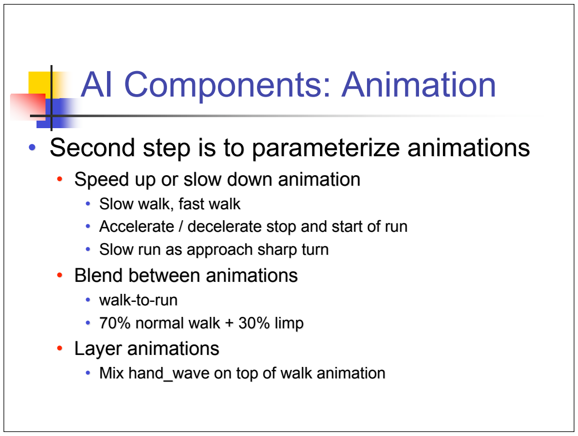
Some animation sequences only take control of part of the body:

- wave_hello
- hand_signal_stop
- swing_ice_axe



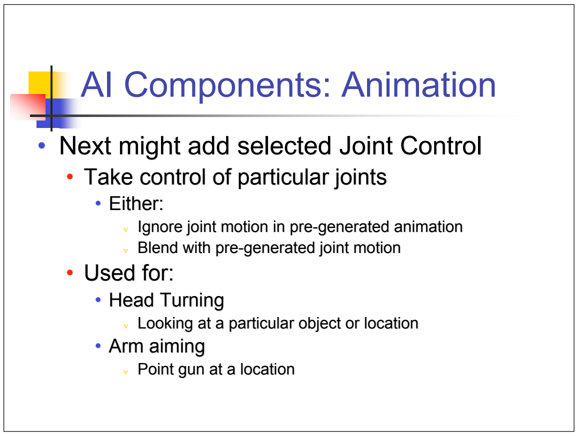
AI Components: Animation

- First step in A.I. is to select which animation sequence or sequences should be applied to a model
- Many influences:
  - Desired behavior chosen by decision system
  - What animation is currently playing
  - The current velocity and direction of the NPC
  - The terrain the NPC is standing on



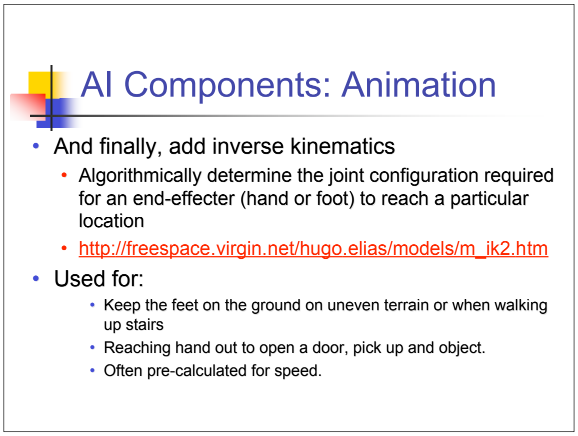
AI Components: Animation

- Second step is to parameterize animations
  - Speed up or slow down animation
    - Slow walk, fast walk
    - Accelerate / decelerate stop and start of run
    - Slow run as approach sharp turn
  - Blend between animations
    - walk-to-run
    - 70% normal walk + 30% limp
  - Layer animations
    - Mix hand_wave on top of walk animation



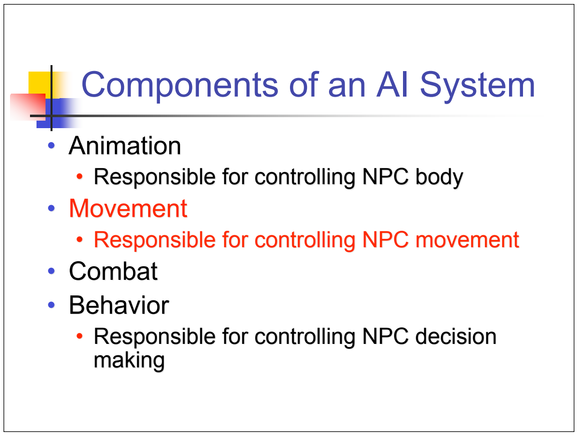
AI Components: Animation

- Next might add selected Joint Control
  - Take control of particular joints
    - Either:
      - Ignore joint motion in pre-generated animation
      - Blend with pre-generated joint motion
  - Used for:
    - Head Turning
      - Looking at a particular object or location
    - Arm aiming
      - Point gun at a location



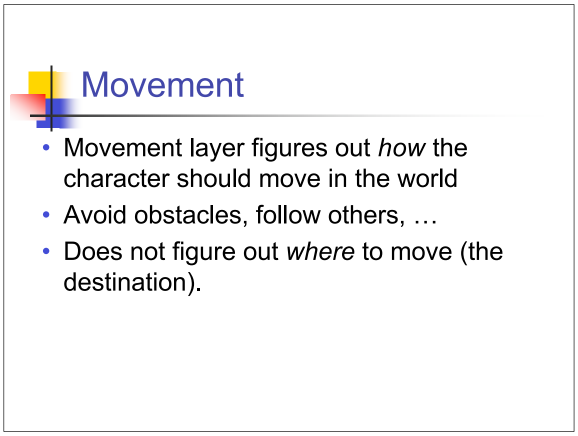
AI Components: Animation

- And finally, add inverse kinematics
  - Algorithmically determine the joint configuration required for an end-effecter (hand or foot) to reach a particular location
  - http://freespace.virgin.net/hugo.elias/models/m_ik2.htm
- Used for:
    - Keep the feet on the ground on uneven terrain or when walking up stairs
    - Reaching hand out to open a door, pick up and object.
    - Often pre-calculated for speed.



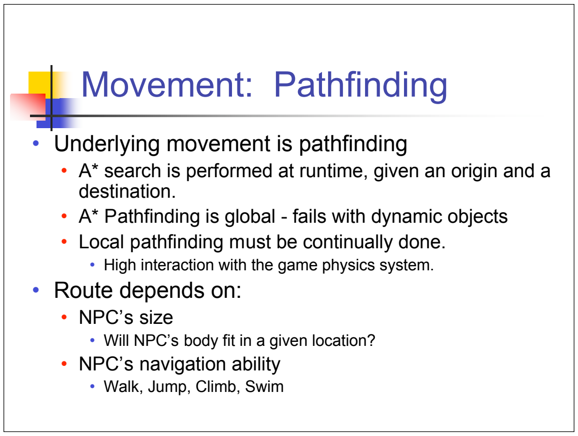
Components of an AI System

- Animation
  - Responsible for controlling NPC body
- Movement
  - Responsible for controlling NPC movement
- Combat
- Behavior
  - Responsible for controlling NPC decision making



Movement

- Movement layer figures out *how* the character should move in the world
- Avoid obstacles, follow others, …
- Does not figure out *where* to move (the destination).



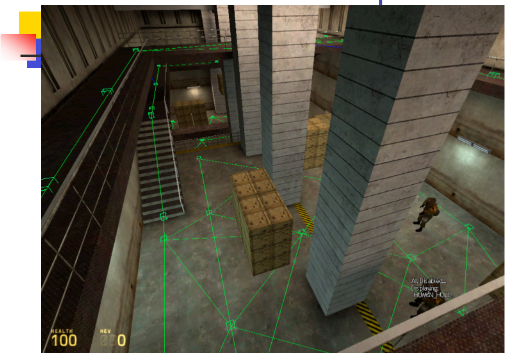
Movement: Pathfinding

- Underlying movement is pathfinding
  - A* search is performed at runtime, given an origin and a destination.
  - A* Pathfinding is global - fails with dynamic objects
  - Local pathfinding must be continually done.
    - High interaction with the game physics system.
- Route depends on:
  - NPC's size
    - Will NPC's body fit in a given location?
  - NPC's navigation ability
    - Walk, Jump, Climb, Swim

Movement: Pathfinding Tools

- Waypoint
  - Position in a map that is used for navigation
  - Usually explicitly placed in world by a level designer
- Link
  - Connection between two waypoints
  - Often annotated with the required navigation type (Jump, Swim, Climb)
  - For a given NPC, two waypoints are linked when:
    - The NPC has room enough to move from one node to another without colliding with the world geometry
    - The NPC has the required navigation ability
- Node Graph
  - Data structure holding all waypoints and links
  - Either generated manually by a level designer or automatically by the computer and annotated by a level designer

Movement: Node Graph



Combat: Most Challenging

- Assessing the situation intelligently
  - Spatial reasoning
- Selecting and executing appropriate tactics
  - Camp, Joust, Circle of Death, Ambush, Flee and Ambush
- Perceptual modeling
  - AI must act in accordance with its perceptions (shouldn't be able to see in dark without night vision goggles, etc.)
- Weapons Combat

Combat: Spatial Reasoning

- 3D map geometry is difficult to parse.
- Solution: Custom databases
  - Place hints throughout the world
  - Can be error-prone and inefficient
  - Does not handle dynamic obstacles

Perceptual Modeling

- Visual subsystem: seeing target
  - Distance to visual stimulus
  - Angle of stimulus relative to field of view
  - Line of sight calculations
- Auditory subsystem
  - Ensure that the AI can hear objects in the world
  - AI must interpret and prioritize sounds
- Tactile subsystem
  - Handles anything the AI can feel
  - Damage notifications and collision notifications

Thief

- Excellent perceptual modelling.
- Auditory & Visual

Weapon Combat

- To-Hit Roll
  - Calculate value to represent the chance to hit, generate random number.
  - If number is above to-hit value, try to miss target.
- Factors:
  - AI skill, Range, Size, Relative Target Velocity, Visibility and Coverage
- Shoot and Miss
  - Pick a target coordinate outside the body
  - Place shot inside target's field of view

Behavior

- Highest-level AI subsystem
- Determines overall behavior, goals, …
- Finite State Machines used to model behavior states.
  - Idle, Patrolling, Combat, Fleeing, Searching, …
- Scripting
  - Pre-defined set of behavioral actions
  - Triggered Events
  - Set AI parameters or send commands to other modules

Quake III Arena

- Released in 1999 by id Software
- Designed to be a multiplayer only game
- The player battles computer-controlled opponents ("bots")
- Bots developed by Jan Paul van Waveren

Quake III Bot AI

- FSM based – Uses a stack for short-term goals
- Use Fuzzy Logic for some decision making
  - Collecting weapons and armor
  - Choosing a weapon for combat
- Fuzzy Relations were generated using Genetic Algorithms
- Each bot has a data file containing weapon preferences and behavior-controlling variables

Data File for 'Bones'

```
//initial weapon weights
#define W_SHOTGUN              750
#define W_MACHINEGUN           10
#define W_GRENADELAUNCHER      10
#define W_ROCKETLAUNCHER       10
#define W_RAILGUN              15
#define W_BFG10K               10
#define W_LIGHTNING            10
#define W_PLASMAGUN            10
```

Bones

Data File for 'Bones'

```
//initial powerup weights
#define W_TELEPORTER           10
#define W_MEDKIT               10
#define W_QUAD                 10
#define W_ENVIRO               400
#define W_HASTE                10
#define W_INVISIBILITY         200
#define W_REGEN                10
#define W_FLIGHT               10
```

Bot Characteristics

- Camper
- Jumper
- Rocket Jumper
- Aggression
- Self-Preservation
- Vengefulness
- Alertness
- Various chat variables

Bot Network

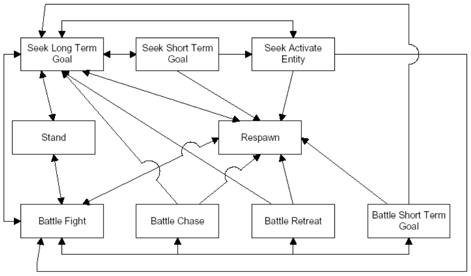


Figure 15.1: AI network.

Quake III Bot Navigation

- AAS(Area Awareness System)
  - Level is subdivided into convex hulls that contain no obstacles
  - Connections between areas are formed

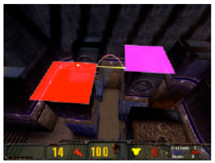


Figure 18.2: jump reachability

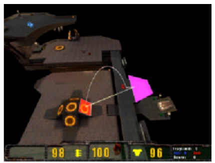


Figure 18.3: jump pad reachability

Bot Chatting

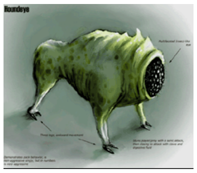
- Deathmatch
  - Not much more than a fun extra
- Team-Play
  - Bots can follow orders to defend, attack, escort
  - Bots will take ‘Team Leader’ position if player doesn’t
  - Team Leader delegates tasks to bots and players

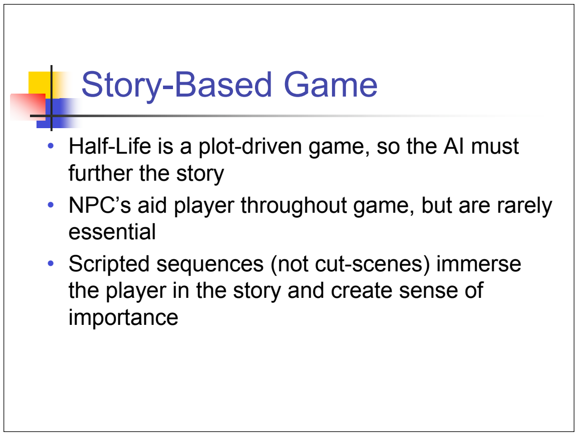
Bot Input

- Bots should simulate human input
  - 90 degree FOV
  - fog and the invisibility powerup impact vision
- Bots use sound to detect enemies

Half-Life

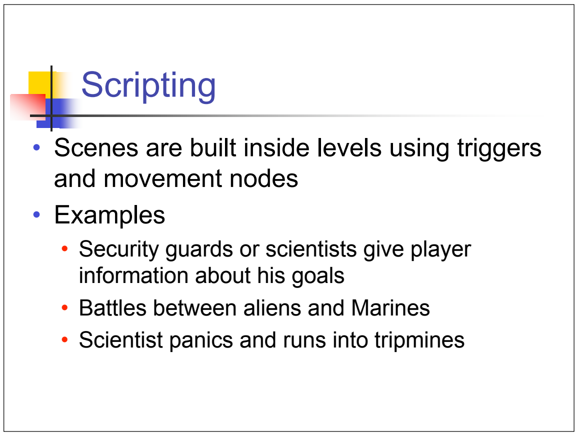
- Released by Valve Software in 1998
- Built using the Quake/Quake 2 engines
- AI uses a “schedule driven state machine”





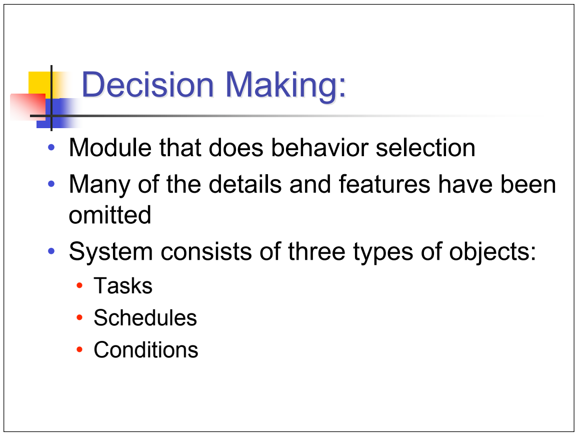
Story-Based Game

- Half-Life is a plot-driven game, so the AI must further the story
- NPC's aid player throughout game, but are rarely essential
- Scripted sequences (not cut-scenes) immerse the player in the story and create sense of importance



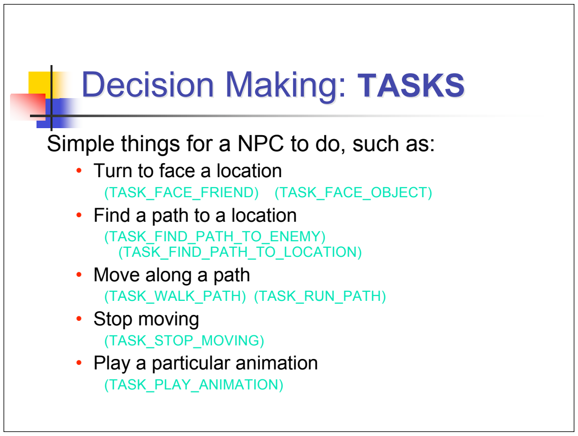
Scripting

- Scenes are built inside levels using triggers and movement nodes
- Examples
  - Security guards or scientists give player information about his goals
  - Battles between aliens and Marines
  - Scientist panics and runs into tripmines



Decision Making:

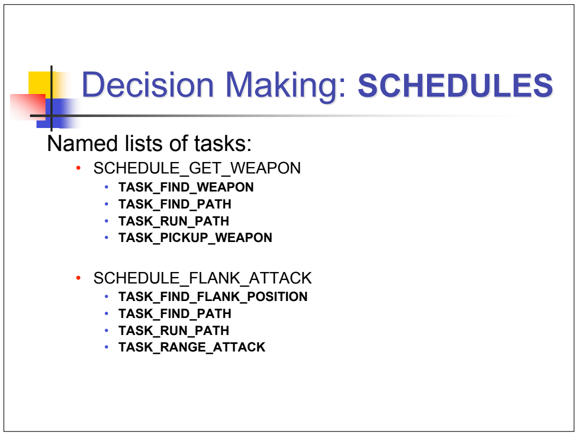
- Module that does behavior selection
- Many of the details and features have been omitted
- System consists of three types of objects:
  - Tasks
  - Schedules
  - Conditions



Decision Making: **TASKS**

Simple things for a NPC to do, such as:

- Turn to face a location
  (TASK_FACE_FRIEND) (TASK_FACE_OBJECT)
- Find a path to a location
  (TASK_FIND_PATH_TO_ENEMY)
  (TASK_FIND_PATH_TO_LOCATION)
- Move along a path
  (TASK_WALK_PATH) (TASK_RUN_PATH)
- Stop moving
  (TASK_STOP_MOVING)
- Play a particular animation
  (TASK_PLAY_ANIMATION)

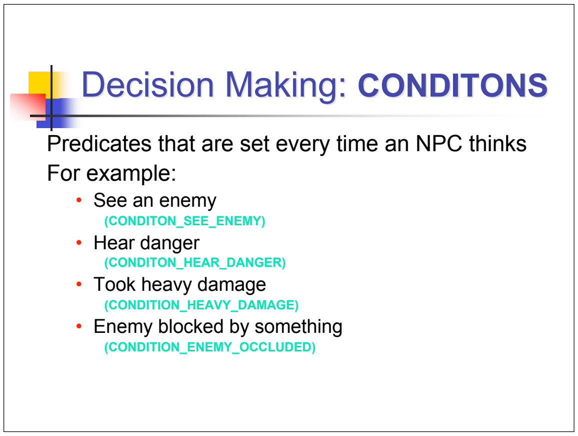


Decision Making: **SCHEDULES**

Named lists of tasks:

- SCHEDULE_GET_WEAPON
  - **TASK_FIND_WEAPON**
  - **TASK_FIND_PATH**
  - **TASK_RUN_PATH**
  - **TASK_PICKUP_WEAPON**

- SCHEDULE_FLANK_ATTACK
  - **TASK_FIND_FLANK_POSITION**
  - **TASK_FIND_PATH**
  - **TASK_RUN_PATH**
  - **TASK_RANGE_ATTACK**



Decision Making: **CONDITONS**

Predicates that are set every time an NPC thinks

For example:

- See an enemy
  (CONDITON_SEE_ENEMY)
- Hear danger
  (CONDITON_HEAR_DANGER)
- Took heavy damage
  (CONDITION_HEAVY_DAMAGE)
- Enemy blocked by something
  (CONDITION_ENEMY_OCCLUDED)

Decision Making:

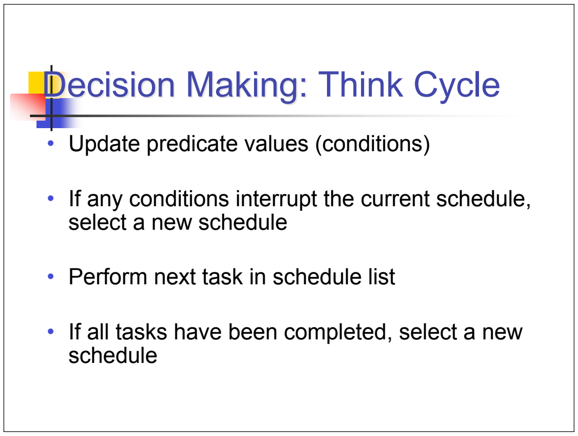
- Conditions serve two purposes:
  - Schedule Selection
  - Schedule Interruption

Decision Making: Conditions

- Used for “rule based” schedule selection
- If (CONDITION_HEAR_DANGER) and not (CONDITION_HAVE_WEAPON)
  select schedule (SCHEDULE_GET_WEAPON)
- If (CONDITION_HAVE_WEAPON) and (CONDITION_OUT_OF_AMMO)
  select schedule (SCHEDULE_RELOAD_WEAPON)

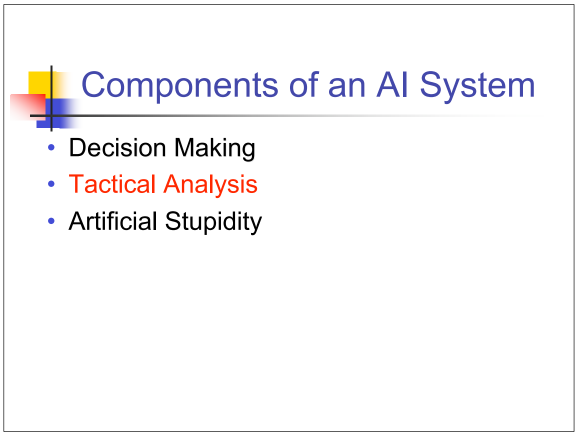
Decision Making: Conditions

- Used for schedule interruption.
- Schedules also contain interrupt conditions.
  - SCHEDULE_GET_WEAPON
    - TASK_FIND_WEAPON
    - TASK_PICKUP_WEAPON
    - CONDITION_HEAVY_DAMAGE
    - CONDITION_ENEMY_DEAD



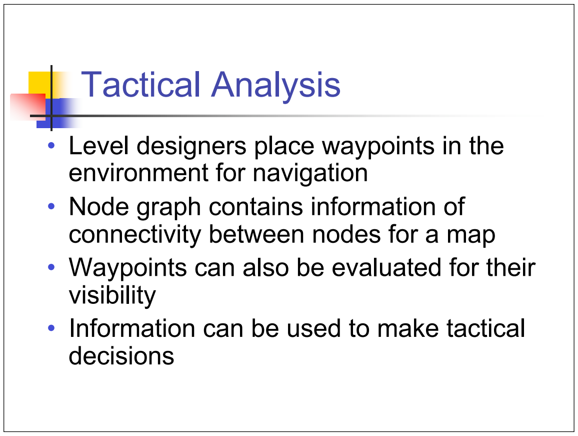
Decision Making: Think Cycle

- Update predicate values (conditions)
- If any conditions interrupt the current schedule, select a new schedule
- Perform next task in schedule list
- If all tasks have been completed, select a new schedule



Components of an AI System

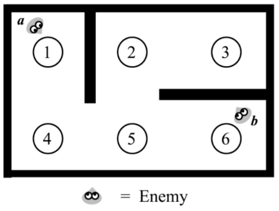
- Decision Making
- Tactical Analysis
- Artificial Stupidity



Tactical Analysis

- Level designers place waypoints in the environment for navigation
- Node graph contains information of connectivity between nodes for a map
- Waypoints can also be evaluated for their visibility
- Information can be used to make tactical decisions

Waypoint Analysis



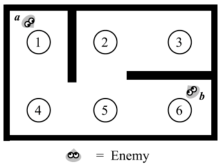
Waypoint Analysis

- Limited CPU time
- Decisions must be made quickly (as few CPU cycles as possible)
- Data must stored efficiently
- Store visibility data in a “bit-string” class

V_a = visibility from node “a”

Waypoint Analysis

		Node					
		1	2	3	4	5	6
(a)	V_1 =	1	0	0	1	0	0
	V_2 =	0	1	1	0	1	0
	V_3 =	0	1	1	0	0	0
	V_4 =	1	0	0	1	1	1
	V_5 =	0	1	0	1	1	1
(b)	V_6 =	0	0	0	1	1	1



Waypoint Analysis

- Danger Nodes
  - Determined by “OR”ing the visibility of all enemy’s (*k*) nearest nodes

$$V := \bigcup_{j=0}^{j=k} V_j$$

- Safe Nodes
  - Is its inverse

$$\overline{V}$$

Waypoint Analysis

DANGER NODES:

$V = V_a \cup V_b = 1\quad 0\quad 0\quad 1\quad 1\quad 1$

Nodes 1, 4, 5 and 6 are dangerous

SAFE NODES:

$\overline{V} = 0\quad 1\quad 1\quad 0\quad 0\quad 0$

Nodes 2 and 3 are safe

$$\begin{array}{r|cccccc|}
 & \multicolumn{6}{c}{\text{Node}} \\
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
(a)\ V_1 = & 1 & 0 & 0 & 1 & 0 & 0 \\
V_2 = & 0 & 1 & 1 & 0 & 1 & 0 \\
V_3 = & 0 & 1 & 1 & 0 & 0 & 0 \\
V_4 = & 1 & 0 & 0 & 1 & 1 & 1 \\
V_5 = & 0 & 1 & 0 & 1 & 1 & 1 \\
(b)\ V_6 = & 0 & 0 & 0 & 1 & 1 & 1 \\
\hline
\end{array}$$

Waypoint Analysis

DANGER NODES:

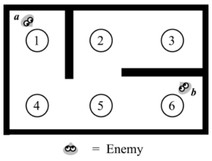
$V = V_a \cup V_b = 1\quad 0\quad 0\quad 1\quad 1\quad 1$

Nodes 1, 4, 5 and 6 are dangerous

SAFE NODES:

$\overline{V} = 0\quad 1\quad 1\quad 0\quad 0\quad 0$

Nodes 2 and 3 are safe



Finding a Safe Attack Position

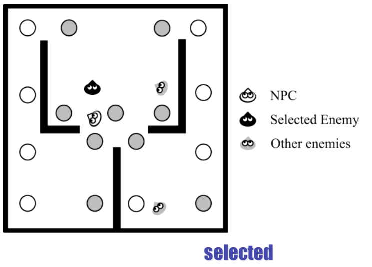
- While attacking a selected enemy, an NPC shouldn’t expose itself to it other enemies
- A good attack position will:
  - Provide line-of-site (LOS) to the selected enemy
  - Provide cover from all other enemies

Finding a Safe Attack Position

- To find such locations, first find all nodes which have LOS to the selected enemy
- Call selected enemy “a”

V_a

Finding a Safe Attack Position

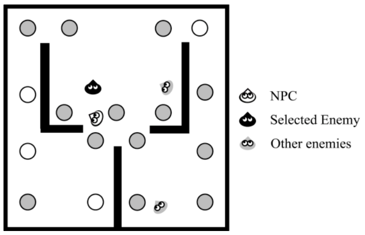


Finding a Safe Attack Position

- Next determine the set of nodes that are visible to all other enemies

$$V_{\overline{a}} = \bigcup_{j=0}^{j=k} V_j, j \neq a$$

Finding a Safe Attack Position



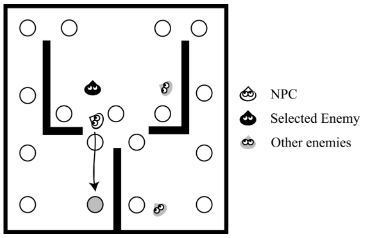
other

Finding a Safe Attack Position

- The set of good attack positions is the set of nodes with LOS to the enemy intersected with the inverse of the set of nodes with LOS to all other enemies

$$V'_a := V_a \,|\, \overline{V}_{\overline{a}}$$

Finding a Safe Attack Position



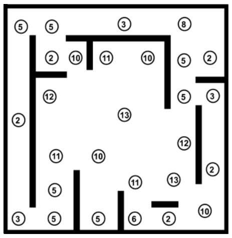
Static Waypoint Evaluation

- Unless cheating is employed, NPCs don’t have full knowledge of the world.
- May not know where all their enemies are located
- Find a good location to wait in for attack
- Not all positions are created equal

Static Waypoint Evaluation

- To find a good set up position:
  - Establish the exposure of all waypoints in a map
  - Process can be done off line, before game is even started

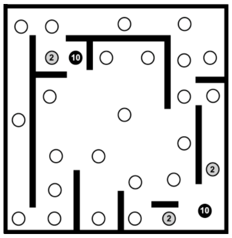
Static Waypoint Evaluation



Static Waypoint Evaluation

- A good location is one which:
  - Has high exposure (visibility)
    - Easy to locate enemies
    - Easy to establish LOS to attack and enemy
  - Has areas of low exposure nearby
    - Can hide easily
    - Can run for cover easily

Static Waypoint Evaluation



Pinch Points

- Observation of human players reveals that experienced players anticipate the actions of their opponents
  - For example, if an enemy enters a room with only a single exit an experienced player will wait just outside the exit setting up an ambush
- Such “pinch points” can be pre-calculated by analyzing the node graph

Pinch Points

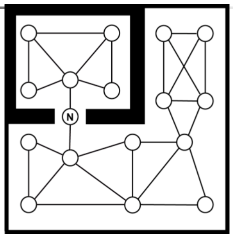
To find pinch points:

For each node, **N** in the node graph with only two neighbors:

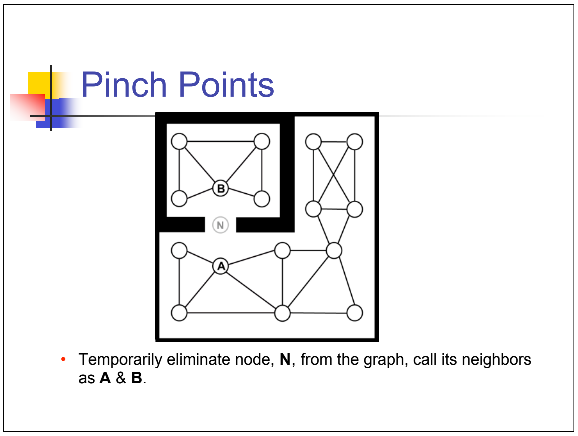
- Temporarily eliminate node, **N**, from the graph, call its neighbors as **A** & **B**.
- If both **A** & **B** are connected to large regions, **N** is not a pinch point, try another **N**.
- Attempt to find a path between **A**& **B**.
- If path exists, **N** is not a pinch point, try another **N**.
- Call the node connected to the larger region, **O** (for outside).
- Call the node connected to the smaller region, **I** (for inside).

Let’s do that again step-by-step:

Pinch Points

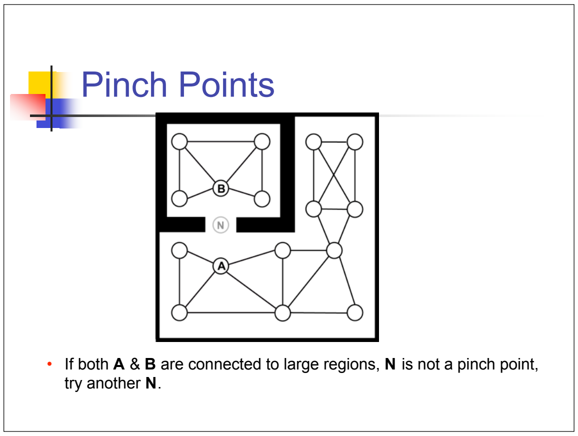


- For each node, **N** in the node graph with only two neighbors:



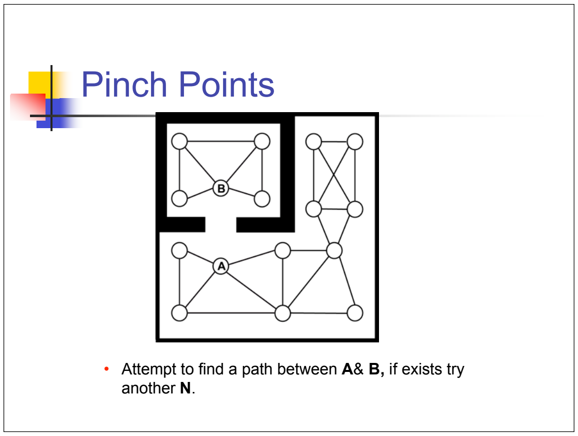
Pinch Points

- Temporarily eliminate node, **N**, from the graph, call its neighbors as **A** & **B**.



Pinch Points

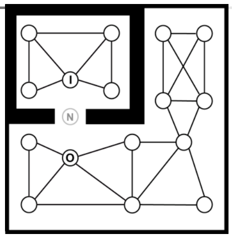
- If both **A** & **B** are connected to large regions, **N** is not a pinch point, try another **N**.



Pinch Points

- Attempt to find a path between **A**& **B,** if exists try another **N**.

Pinch Points



- Call the node connected to the larger region, **O** (for outside).
- Call the node connected to the smaller region, **I** (for inside).

Pinch Points

Once a pinch point has been located a good ambush location is one which:

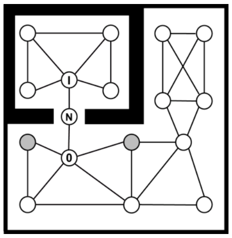
- Has a line of site to the waypoint outside the pinch location “O”
- Can’t be seen from the pinch location “N”

Pinch Points

- Nodes that have a line of site to pinch location “O” V_O
- Can’t be seen from the pinch location “N” $\overline{V}_N$
- Good ambush locations is their intersection:

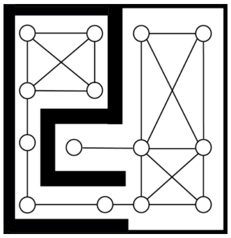
$$V_P = V_O \mid \overline{V}_N$$

Pinch Points



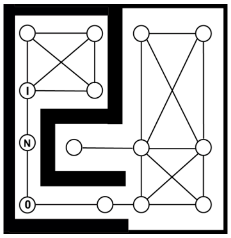
Pinch Points

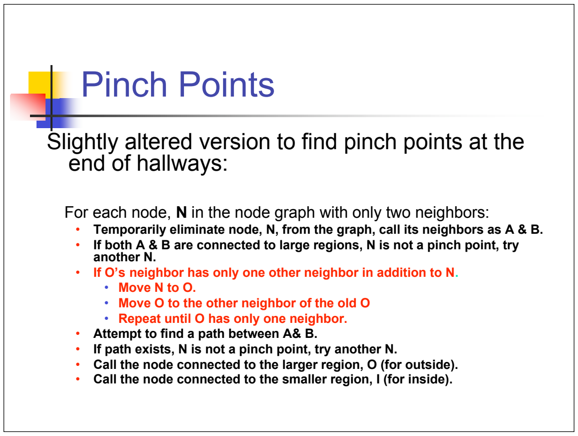
Another Example:



Pinch Points

Result:



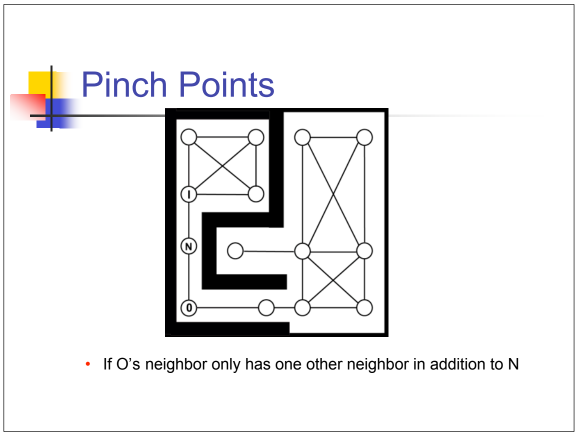


Pinch Points

Slightly altered version to find pinch points at the end of hallways:

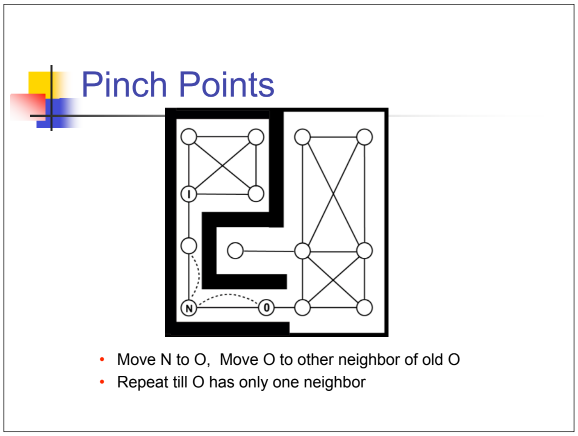
For each node, **N** in the node graph with only two neighbors:

- **Temporarily eliminate node, N, from the graph, call its neighbors as A & B.**
- **If both A & B are connected to large regions, N is not a pinch point, try another N.**
- **If O's neighbor has only one other neighbor in addition to N.**
  - **Move N to O.**
  - **Move O to the other neighbor of the old O**
  - **Repeat until O has only one neighbor.**
- **Attempt to find a path between A& B.**
- **If path exists, N is not a pinch point, try another N.**
- **Call the node connected to the larger region, O (for outside).**
- **Call the node connected to the smaller region, I (for inside).**



Pinch Points

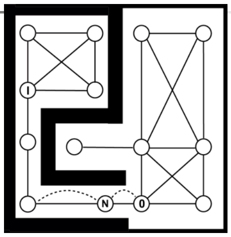
- If O's neighbor only has one other neighbor in addition to N



Pinch Points

- Move N to O, Move O to other neighbor of old O
- Repeat till O has only one neighbor

Pinch Points



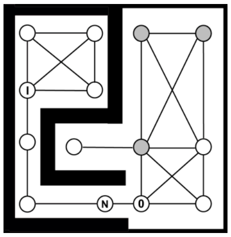
- Move N to O, Move O to other neighbor of old O
- Repeat till O has only one neighbor

Pinch Points

- Calculate good ambush locations:

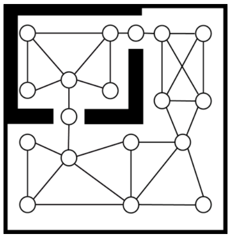
$$V_P = V_O \,|\, \overline{V_N}$$

Pinch Points

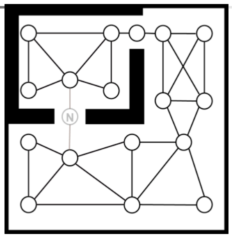


Pinch Points

Final Example:

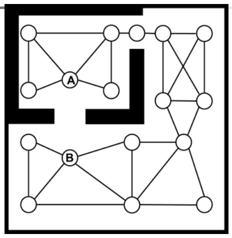


Pinch Points

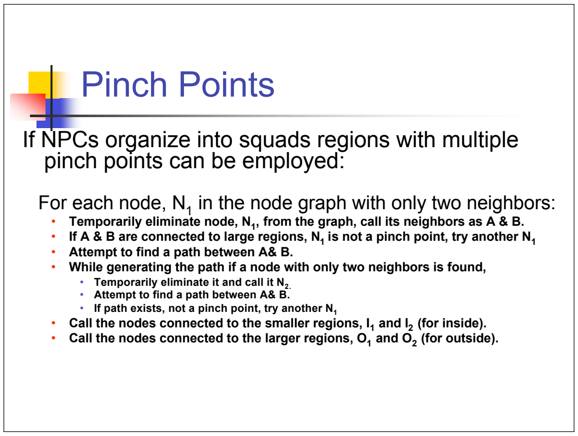


- For each node, **N** in the node graph with only two neighbors

Pinch Points



- **Attempt to find a path between A& B.**
- **If path exists, N is not a pinch point, try another N**

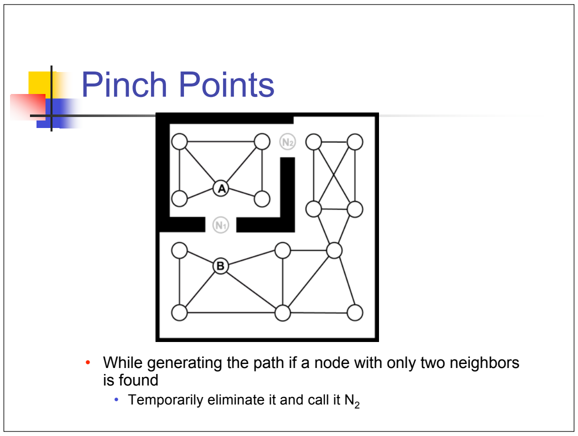


Pinch Points

If NPCs organize into squads regions with multiple pinch points can be employed:

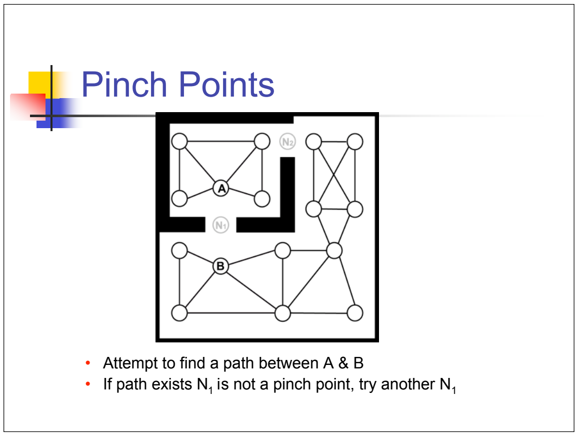
For each node, N_1 in the node graph with only two neighbors:

- Temporarily eliminate node, N_1, from the graph, call its neighbors as A & B.
- If A & B are connected to large regions, N_1 is not a pinch point, try another N_1
- Attempt to find a path between A& B.
- While generating the path if a node with only two neighbors is found,
  - Temporarily eliminate it and call it N_2.
  - Attempt to find a path between A& B.
  - If path exists, not a pinch point, try another N_1
- Call the nodes connected to the smaller regions, I_1 and I_2 (for inside).
- Call the nodes connected to the larger regions, O_1 and O_2 (for outside).



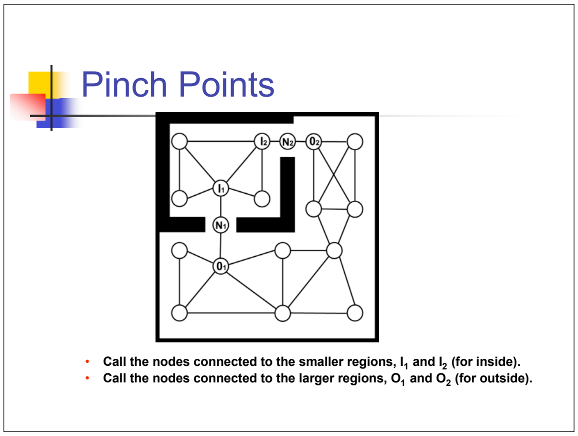
Pinch Points

- While generating the path if a node with only two neighbors is found
  - Temporarily eliminate it and call it N_2



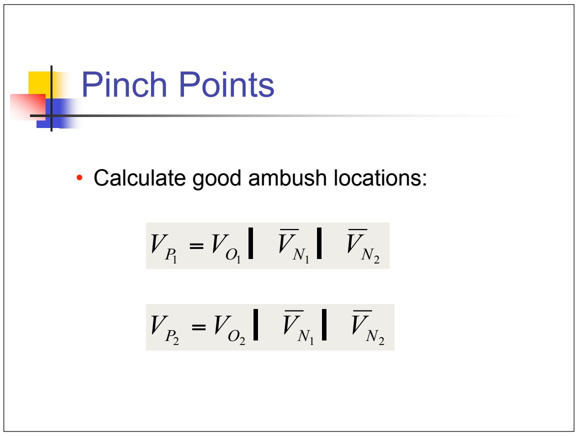
Pinch Points

- Attempt to find a path between A & B
- If path exists N_1 is not a pinch point, try another N_1



Pinch Points

- Call the nodes connected to the smaller regions, I_1 and I_2 (for inside).
- Call the nodes connected to the larger regions, O_1 and O_2 (for outside).

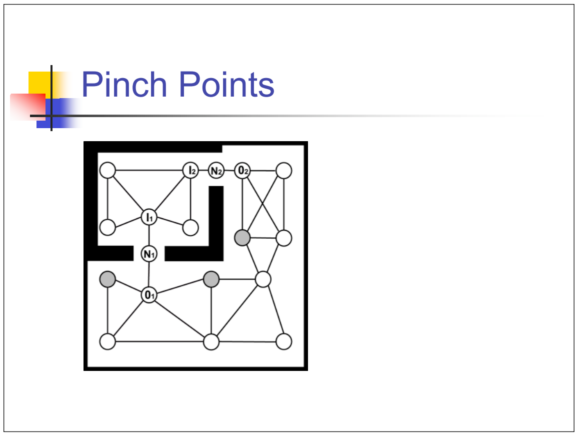


Pinch Points

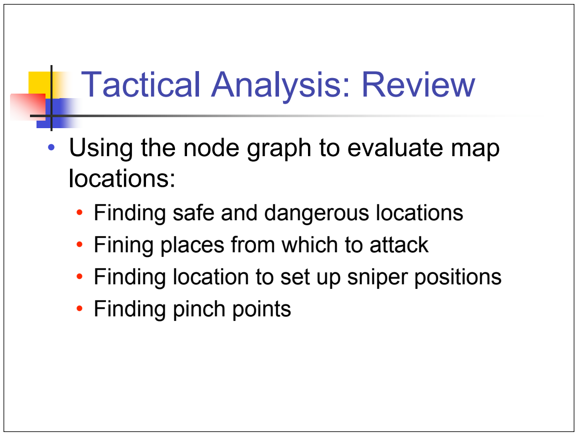
- Calculate good ambush locations:

$$V_{P_1} = V_{O_1} \mid \overline{V}_{N_1} \mid \overline{V}_{N_2}$$

$$V_{P_2} = V_{O_2} \mid \overline{V}_{N_1} \mid \overline{V}_{N_2}$$

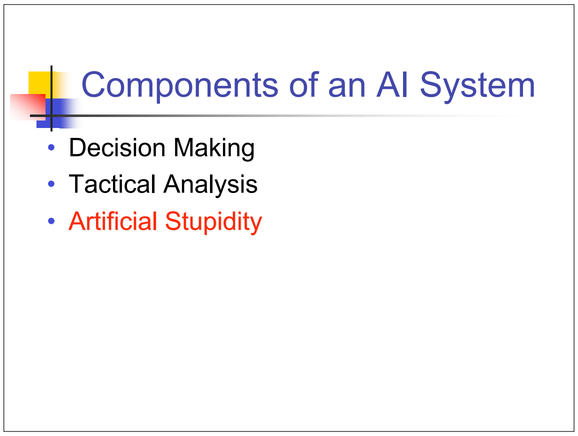


Pinch Points



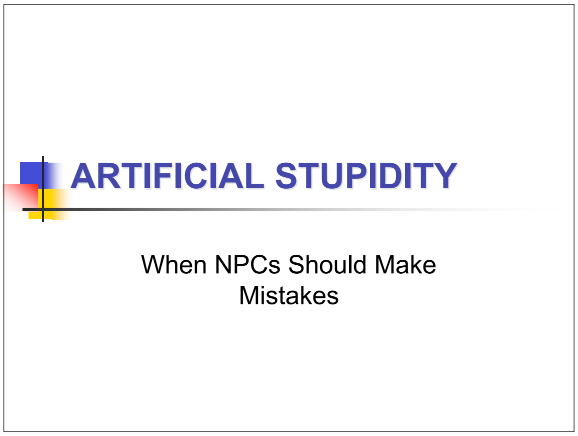
Tactical Analysis: Review

- Using the node graph to evaluate map locations:
  - Finding safe and dangerous locations
  - Fining places from which to attack
  - Finding location to set up sniper positions
  - Finding pinch points



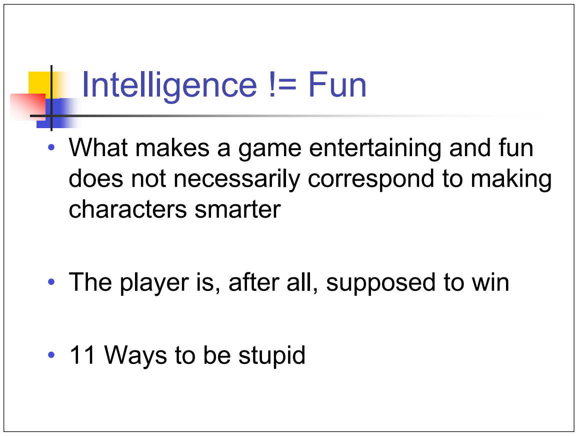
Components of an AI System

- Decision Making
- Tactical Analysis
- Artificial Stupidity



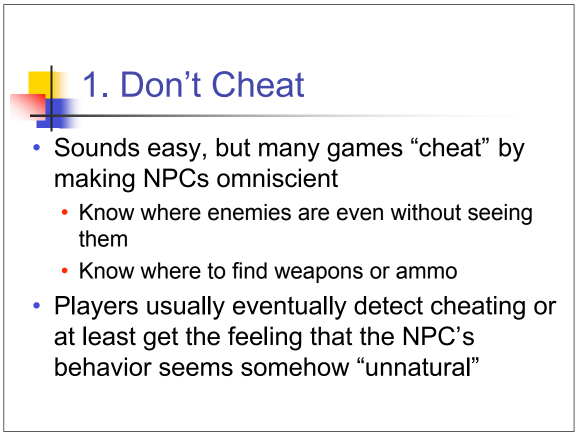
ARTIFICIAL STUPIDITY

When NPCs Should Make
Mistakes



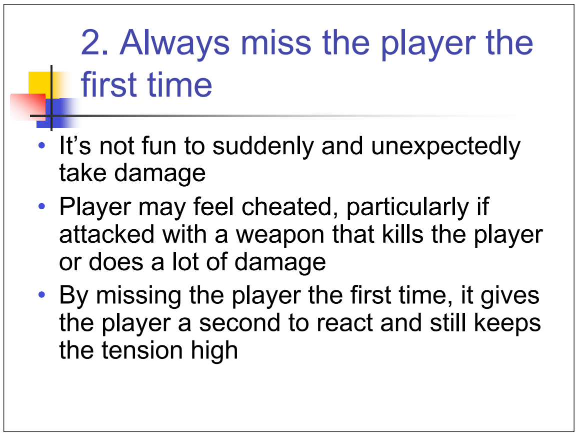
Intelligence != Fun

- What makes a game entertaining and fun does not necessarily correspond to making characters smarter
- The player is, after all, supposed to win
- 11 Ways to be stupid



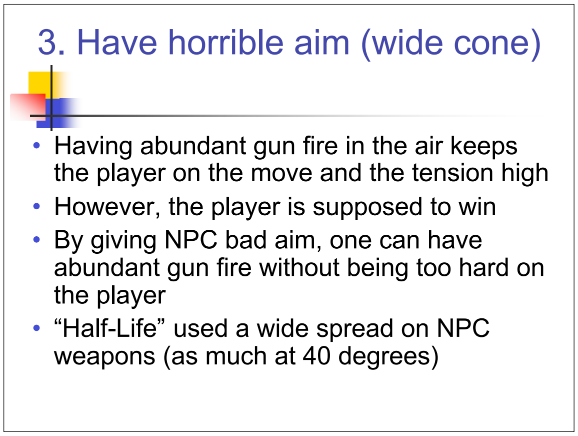
1. Don’t Cheat

- Sounds easy, but many games “cheat” by making NPCs omniscient
  - Know where enemies are even without seeing them
  - Know where to find weapons or ammo
- Players usually eventually detect cheating or at least get the feeling that the NPC’s behavior seems somehow “unnatural”



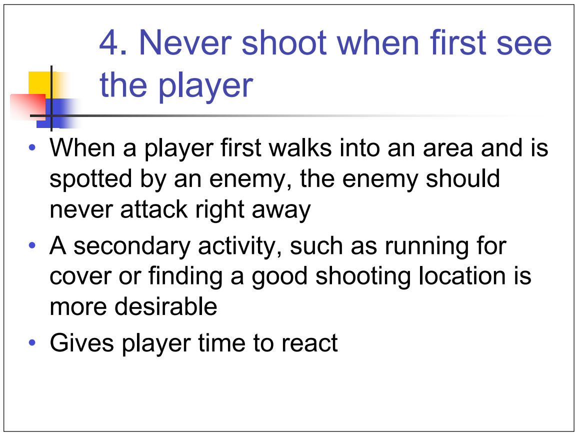
2. Always miss the player the first time

- It’s not fun to suddenly and unexpectedly take damage
- Player may feel cheated, particularly if attacked with a weapon that kills the player or does a lot of damage
- By missing the player the first time, it gives the player a second to react and still keeps the tension high



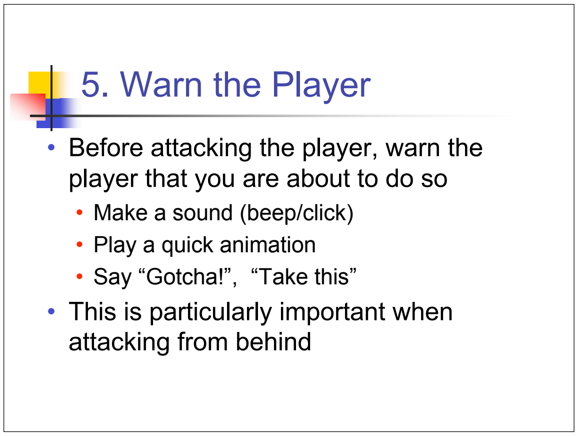
3. Have horrible aim (wide cone)

- Having abundant gun fire in the air keeps the player on the move and the tension high
- However, the player is supposed to win
- By giving NPC bad aim, one can have abundant gun fire without being too hard on the player
- “Half-Life” used a wide spread on NPC weapons (as much at 40 degrees)



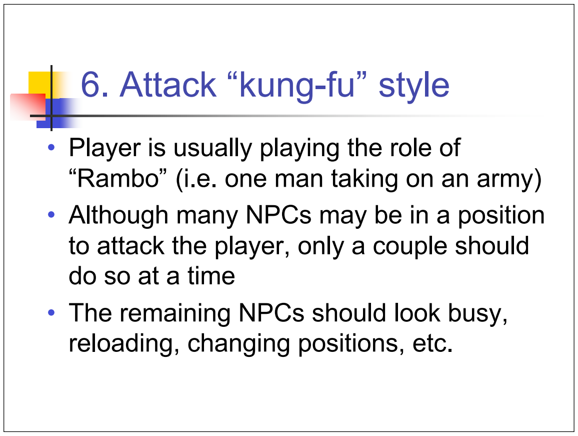
4. Never shoot when first see the player

- When a player first walks into an area and is spotted by an enemy, the enemy should never attack right away
- A secondary activity, such as running for cover or finding a good shooting location is more desirable
- Gives player time to react



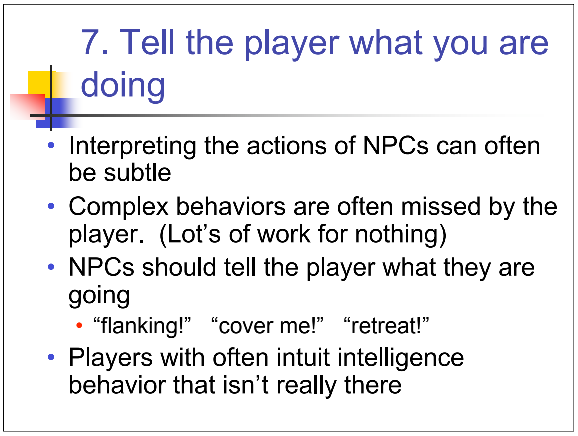
5. Warn the Player

- Before attacking the player, warn the player that you are about to do so
  - Make a sound (beep/click)
  - Play a quick animation
  - Say “Gotcha!”, “Take this”
- This is particularly important when attacking from behind



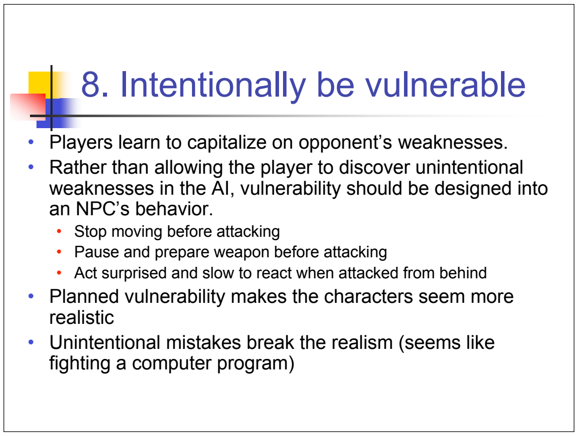
6. Attack “kung-fu” style

- Player is usually playing the role of “Rambo” (i.e. one man taking on an army)
- Although many NPCs may be in a position to attack the player, only a couple should do so at a time
- The remaining NPCs should look busy, reloading, changing positions, etc.



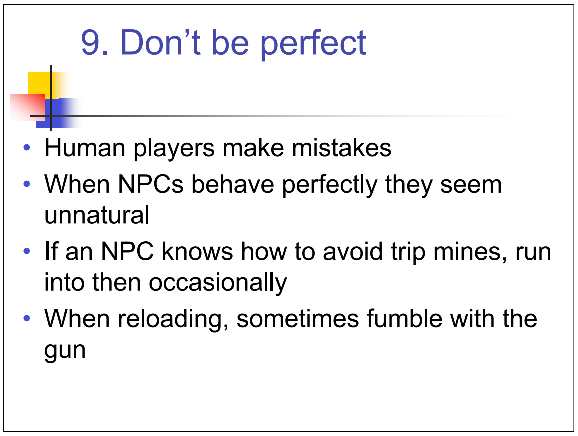
7. Tell the player what you are doing

- Interpreting the actions of NPCs can often be subtle
- Complex behaviors are often missed by the player. (Lot’s of work for nothing)
- NPCs should tell the player what they are going
  - “flanking!” “cover me!” “retreat!”
- Players with often intuit intelligence behavior that isn’t really there



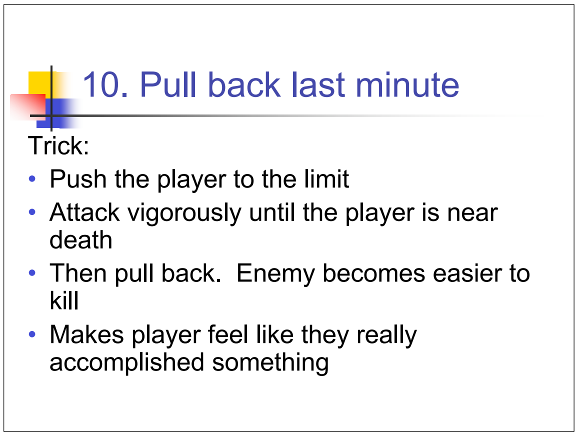
8. Intentionally be vulnerable

- Players learn to capitalize on opponent’s weaknesses.
- Rather than allowing the player to discover unintentional weaknesses in the AI, vulnerability should be designed into an NPC’s behavior.
  - Stop moving before attacking
  - Pause and prepare weapon before attacking
  - Act surprised and slow to react when attacked from behind
- Planned vulnerability makes the characters seem more realistic
- Unintentional mistakes break the realism (seems like fighting a computer program)



9. Don’t be perfect

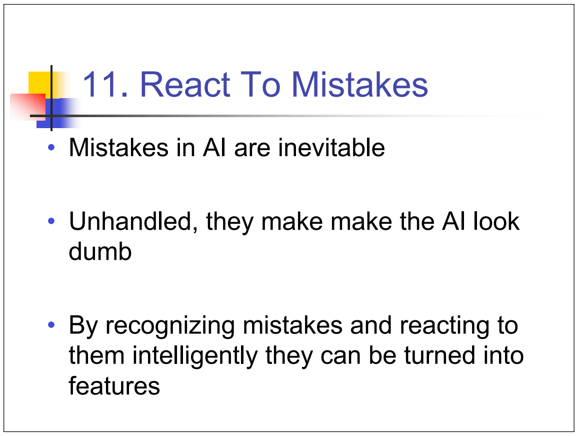
- Human players make mistakes
- When NPCs behave perfectly they seem unnatural
- If an NPC knows how to avoid trip mines, run into then occasionally
- When reloading, sometimes fumble with the gun



10. Pull back last minute

Trick:

- Push the player to the limit
- Attack vigorously until the player is near death
- Then pull back. Enemy becomes easier to kill
- Makes player feel like they really accomplished something



11. React To Mistakes

- Mistakes in AI are inevitable
- Unhandled, they make make the AI look dumb
- By recognizing mistakes and reacting to them intelligently they can be turned into features

11. React To Mistakes

- Example 1:
  - Occasionally when an NPC throws a grenade, it bounces of another object and lands back at the NPCs feet
    - (Note that the player occasionally makes this mistake too!)
  - Looks dumb as the NPC blows himself up
  - If the NPC reacts, however, the mistake turns into a feature:
    - NPC body and facial expression can show surprise, fear
    - NPC can say “Oh Shoot!” or “Doh!”

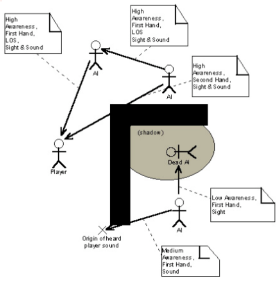
11. React To Mistakes

- Example 2:
  - Player throws a grenade at a group of NPCs. As they are crowded together not all of them are able to find a path to get away
  - Looks dumb if the NPCs that can’t get away, shuffle around trying to get out
  - If we detect that the problem has arisen, can have the trapped NPC’s react
    - Crouch down and put hands over head

Thief

- Developer - Looking Glass
- Publisher - Eidos interactive
- Revolutionary “Dark Engine"
- Based on stealth
- Released November 11th, 1998
- First person, though newest sequel allow 3PS.

“Dark Engine”



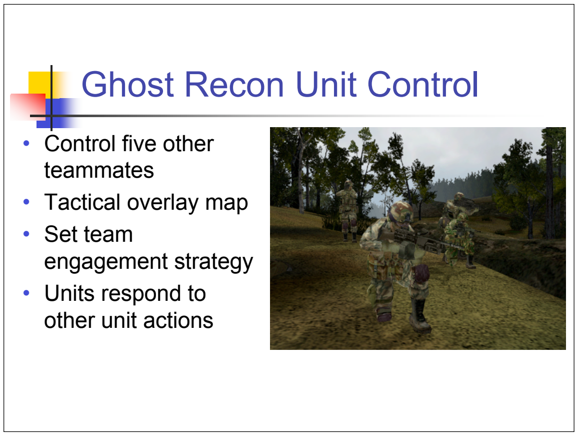
Lightly scripted game
Specifically single-player
Multi-state sense system
Decision state machines
Centers around the system's output

Tom Clancy's GHOST RECON

- Published - Ubi Soft Entertainment
- Greg Stelmack, lead engineer
- Development - Red Storm Entertainment
- Realistic combat battlefield game
- Released November 13th, 2001

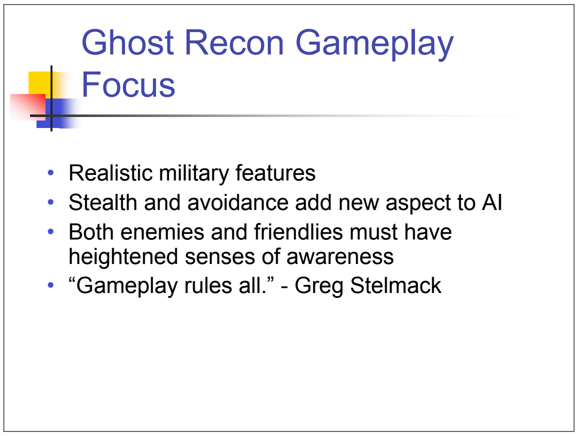
Ghost Recon AI Technique

- A lot of scripting for individual missions
- Enemy and team units use FSM’s
- Modifiable hierarchical commands
- Local navigation and pathfinding (causes some hang-ups small environmental details)



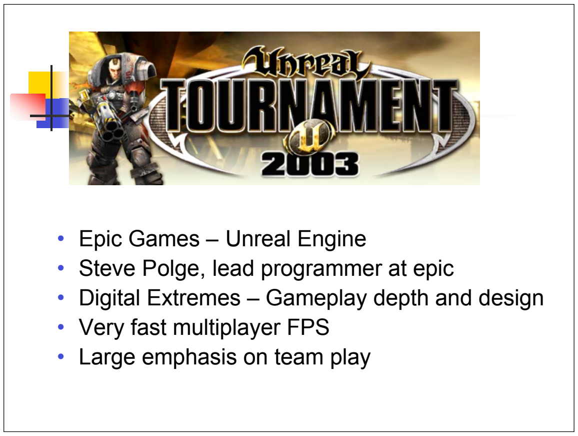
Ghost Recon Unit Control

- Control five other teammates
- Tactical overlay map
- Set team engagement strategy
- Units respond to other unit actions



Ghost Recon Gameplay Focus

- Realistic military features
- Stealth and avoidance add new aspect to AI
- Both enemies and friendlies must have heightened senses of awareness
- “Gameplay rules all.” - Greg Stelmack


Unreal
TOURNAMENT
2003

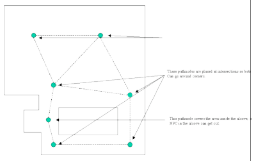
- Epic Games – Unreal Engine
- Steve Polge, lead programmer at epic
- Digital Extremes – Gameplay depth and design
- Very fast multiplayer FPS
- Large emphasis on team play

Unreal Scripting

- UnrealScript much like Java/C++
- Scripting used to control specific Bot actions
- Every respawned Bot checks script flag

Unreal Pathfinding

- **B**ased upon the common pathnode-using technique for navigation
- Uses a pre-computed data structure for guiding movement
- Complex algorithm-controlled assemblage of linked lists, Navigation Points, and Binary Space Partitioning (BSP) collision data


These pathnodes are placed at intersections so bot
Can go around corners.
This pathnode covers the area inside the alcove,
NPC in the alcove can get out.

Unreal Bot Combat

- AI uses states heavily
- Several triggers that determine Bot’s actions
- “Type” of Bot determines fighting style
- Accuracy and speed factor into Bot’s difficulty level
- Fun factor heavily influences Bot strategy

Unreal Team Play

- Incorporates several team oriented games:
- -Team Deathmatch
- - Capture the flag
- - Bombing run
- - Double domination
- Hierarchical AI system
- Player controlled team
- Bots have numerous types of flocking patterns
- Team bots are mediocre, while enemy bots are excellent

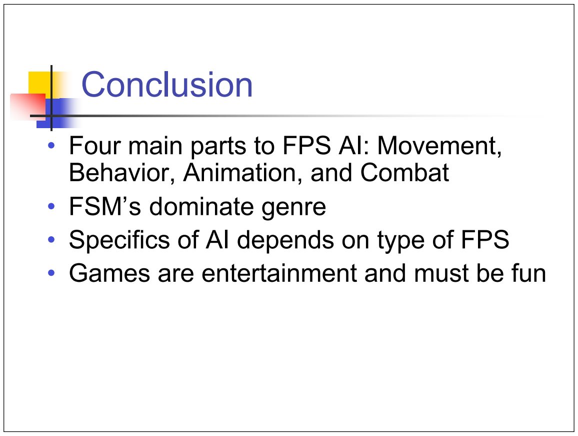
VBS

VBS1™ VIRTUAL BATTLEFIELD SYSTEMS

- Fully interactive, three-dimensional training system
- Photo-realistic terrain
- Bohemia interactive studios
- Flashpoint engine includes proprietary AI algorithms

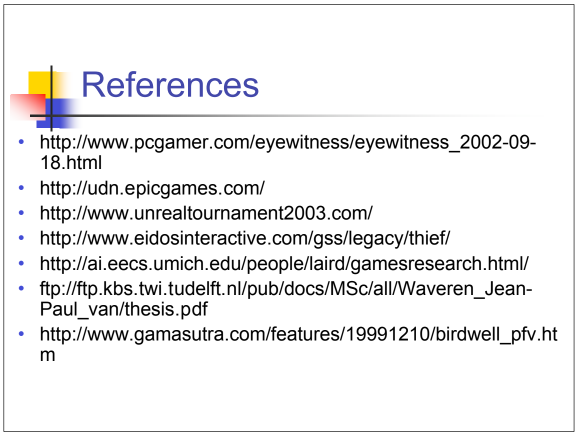
Promising Future FPS

- Halo 2
- Deus Ex 2
- Doom 3



Conclusion

- Four main parts to FPS AI: Movement, Behavior, Animation, and Combat
- FSM’s dominate genre
- Specifics of AI depends on type of FPS
- Games are entertainment and must be fun



References

- http://www.pcgamer.com/eyewitness/eyewitness_2002-09-18.html
- http://udn.epicgames.com/
- http://www.unrealtournament2003.com/
- http://www.eidosinteractive.com/gss/legacy/thief/
- http://ai.eecs.umich.edu/people/laird/gamesresearch.html/
- ftp://ftp.kbs.twi.tudelft.nl/pub/docs/MSc/all/Waveren_Jean-Paul_van/thesis.pdf
- http://www.gamasutra.com/features/19991210/birdwell_pfv.htm