Artificial Intelligence

CSCI4120 Principle of Computer Game Software

Overview

- Computer games are to create a sense of real world in many cases
- Making opponents more human like => Use of AI in games
- Creating the illusion of a world similar as ours => collision detection & response, physics

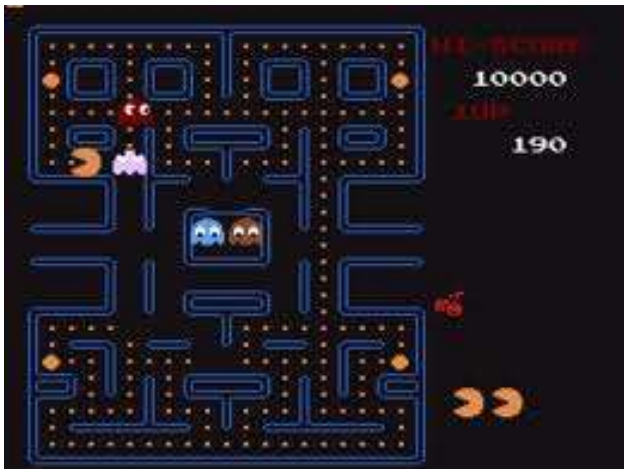
Artificial intelligence

- *If 1% of your art assets are not perfect, not too many people notice.*
  If 1% of your audio assets are not perfect, it doesn't hurt too much.
  But, if 1% of your AI is off, it can potentially affect the entire game.

- Important game play value by making game challenging, addictive
- Computer simulation of intelligent behavior
- But what is "*intelligence*"?
  - Behavior close to human
  - Human did stupid things also!

History

- Typically no intelligence built in for early game such as **Pacman**
- Same movement pattern regardless of player's movement - due to limitations in computational power



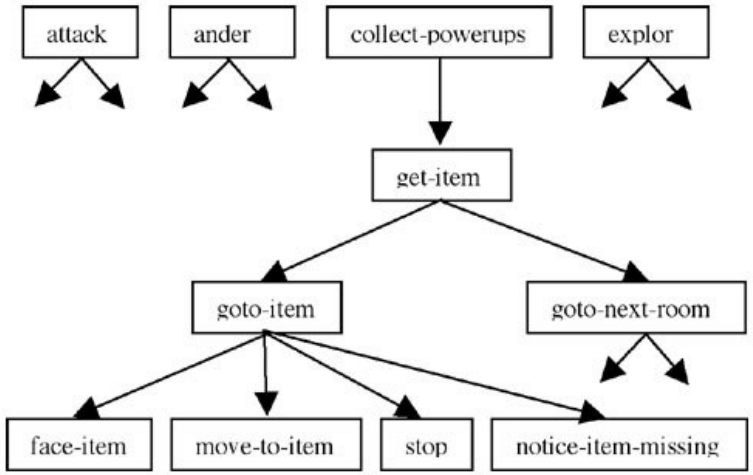
Pacman, ghosts were not chasing, just wandering (1980)

- Started usage in early turn-based strategy game such as **Civilization** & **SanGoKu**
- Opponents will play with tactics, which create much fun

Civilization (1991)

History

- New *FPS* game such as **Halflife** with tactical AI as well as **QuakeBot** for deathmatch play



AI for QuakeBot (1996)

- **Sims** use Fuzzy Finite State Machine to model its artificial lives

- **Black & White** adopt the concept of machine learning

The Sims(2000)

Before founding Deepmind, Demis Hassabis is the lead AI programmer at Lionhead Studios

History

- Left 4 Dead pioneering the use of AI in controlling flow of a cooperative multiplayer game

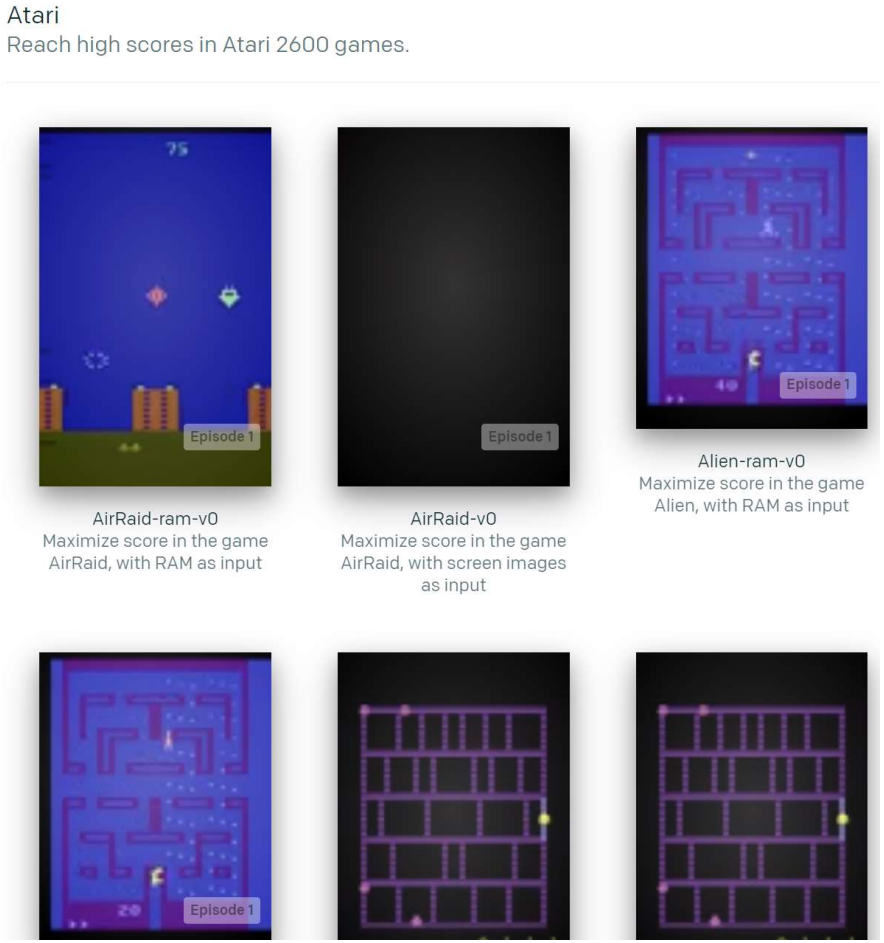
Left 4 Dead (2008)

- **The Division** features automated moving between cover for both players and NPC

The Divisions (2016)

History

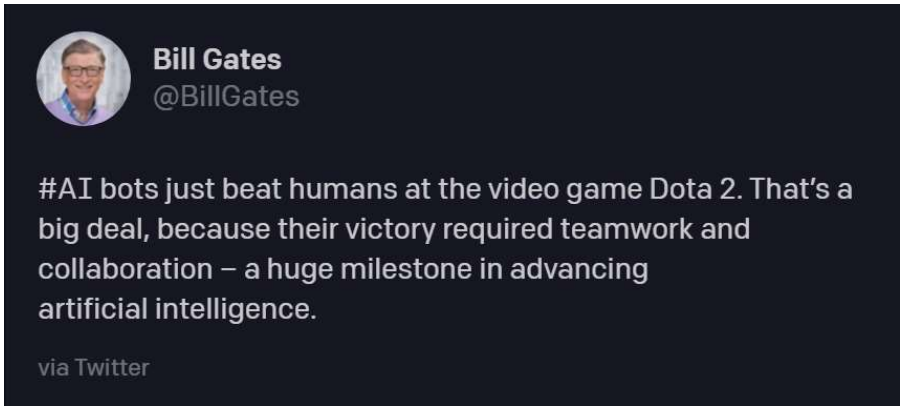
- OpenAI demonstrates using reinforcement learning to train computer agent to play a number of Atari classic games (2017)
- This involves the AI try to figure a way to improve its gaming actions by learning from the result of a move



https://gym.openai.com/envs/#atari

History

- Traditional AI cannot play modern complex game as good as that of professional human player
- OpenAI's OpenAI Five beat 2018 Dota2 esport champion team in 2019 series of games
- It learnt by playing over 10,000 years of games against itself



AI systems

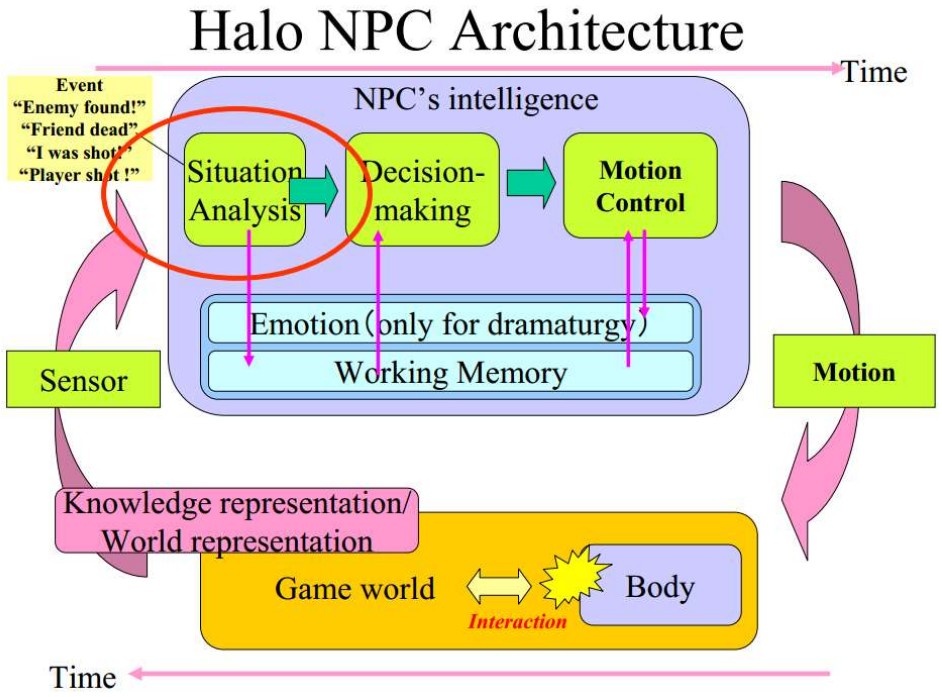
- Come in two favors:

1. *Entities* (virtual characters, action AI) in game world e.g. enemies, non-playable characters(NPC)
2. *Abstract controllers* (strategy AI)– collection of routines that provide group dynamics to overall system

Action AI

- Four elements

1. A sensor or input system
2. A working memory
3. Reasoning/analysis core
4. Action/output system

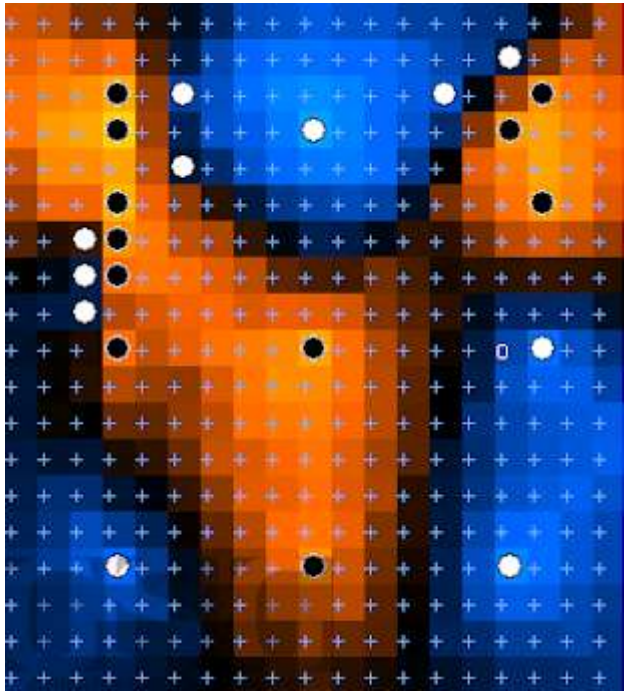


Sensing the world

- To make the computer able to see, hear, or feel
- Use the world information for reasoning/analysis
- Individual AI(e.g. enemies in typical action game):
  1. Where is the player – easy
  2. Geometry of surrounding – difficult in 3D, costly to collect, analyze
  3. Which weapon it & player using - easy

Sensing the world

- master controller e.g. AOE(strategy)
  1. Balance of power in subarea of map
  2. How much of resources currently have
  3. Breakdown of unit type
  4. Status in technology tree



• , ⁰ : units
Colored regions : different forces
Black borders: frontlines

Sensing the world

- Path computations is heavily used e.g.
  1. find a path for a unit to navigate through the map
  2. Advance to a point in technology tree so as to be able to produce nuclear weapon
- Computing balance of power is too complicated that re-computation performed once every N frames

Action Game Implementation

- A monster will patrol or stand & looking for enemy, e.g. patrol code

Walk it's beat

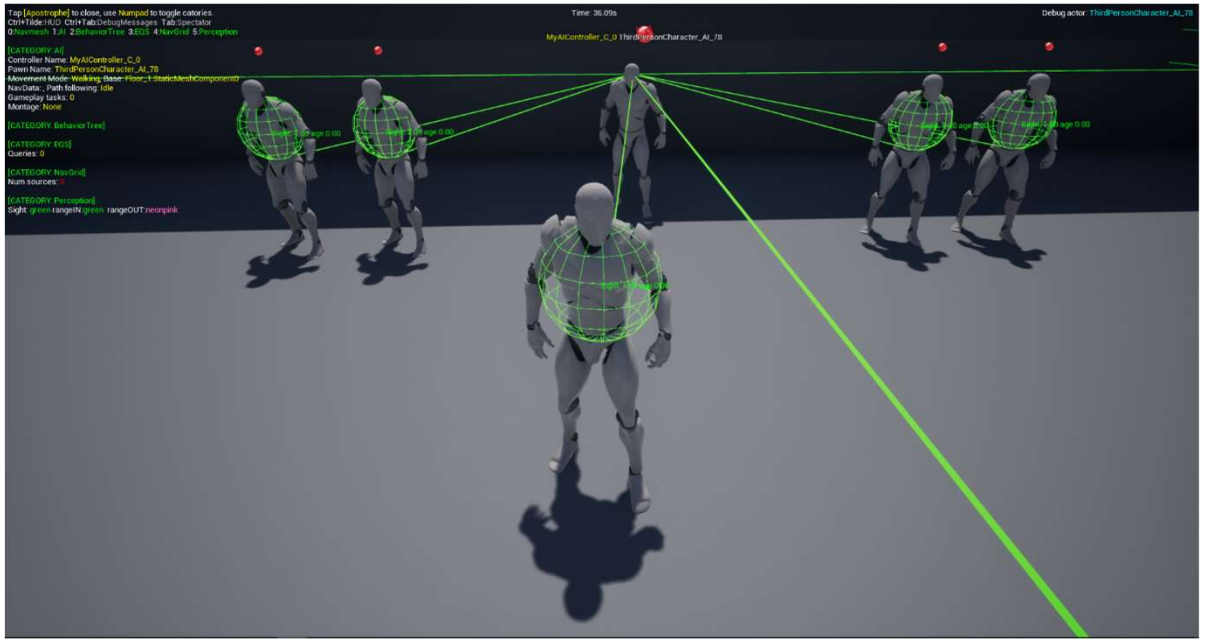
```
M_MoveToGoal (self, dist);

// check for noticing a player
if (FindTarget (self))
    return;
```

Terminate current task(patrol)

Note the above actions are performed on per frame basis

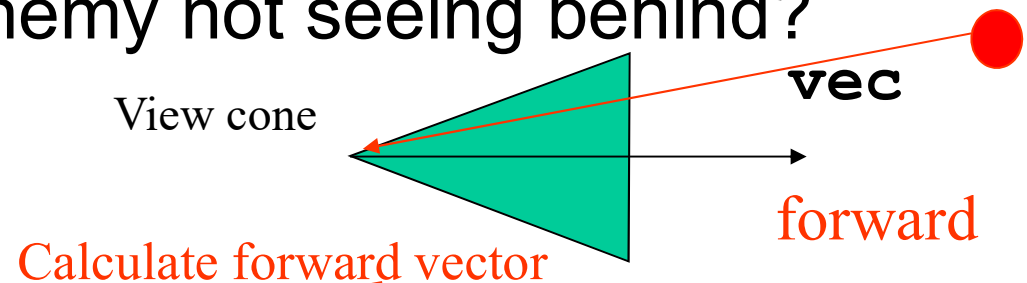
Find Target

- Try to "*see*" or locate an enemy
- Some kinds of vision emulation
- e.g. when a player fires a missile, the point of impact becomes a fake player so that monsters that see the impact will respond as if they had seen the player.



Vision Implementation

- To make an AI individual “sees”, first define a view cone
- How to make the enemy not seeing behind?




```
vec3_t vec, forward;
float dot;

AngleVectors (self->s.angles, forward, NULL, NULL);
VectorSubtract (other->s.origin, self->s.origin, vec);
VectorNormalize (vec);
dot = DotProduct (vec, forward);
if (dot > 0.3) return true;
return false;
```

Calculate forward vector

Calculate vec

Field of view (view angle of the unit)

Further visibility

- If any objects pass the previous *infront()* test, range check for whether close enough

```
vec3_t      v;                  float   len;

VectorSubtract (self->s.origin, other->s.origin, v);
len = VectorLength (v);
if (len < MELEE_DISTANCE)
      return RANGE_MELEE;
if (len < 500)
      return RANGE_NEAR;
if (len < 1000)
      return RANGE_MID;
return RANGE_FAR;
```

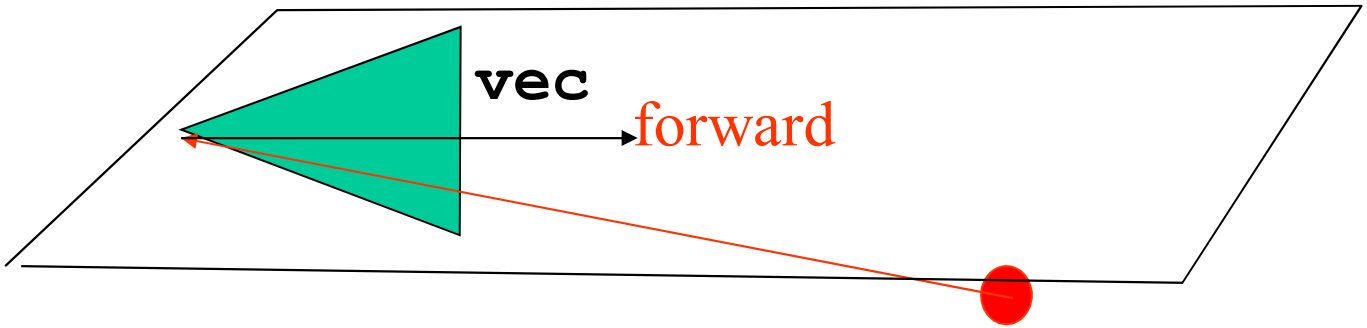

Further visibility

- Further visibility tests such as objects in between are tested, e.g. if we want the AI entity can only see things in the light

```
local entity e,targ;

targ = self.enemy;
e = findradius (targ.origin,200);
if (e.classname == "light")
   nearlight = TRUE;
if (!(nearlight) && (random() < 0.7))
   return;
```


Drawback

- Assume the world is all on same layer(level)
- Break down when player and computer AI are on different levels



- Overall simple enough to be implemented real time

Chasing

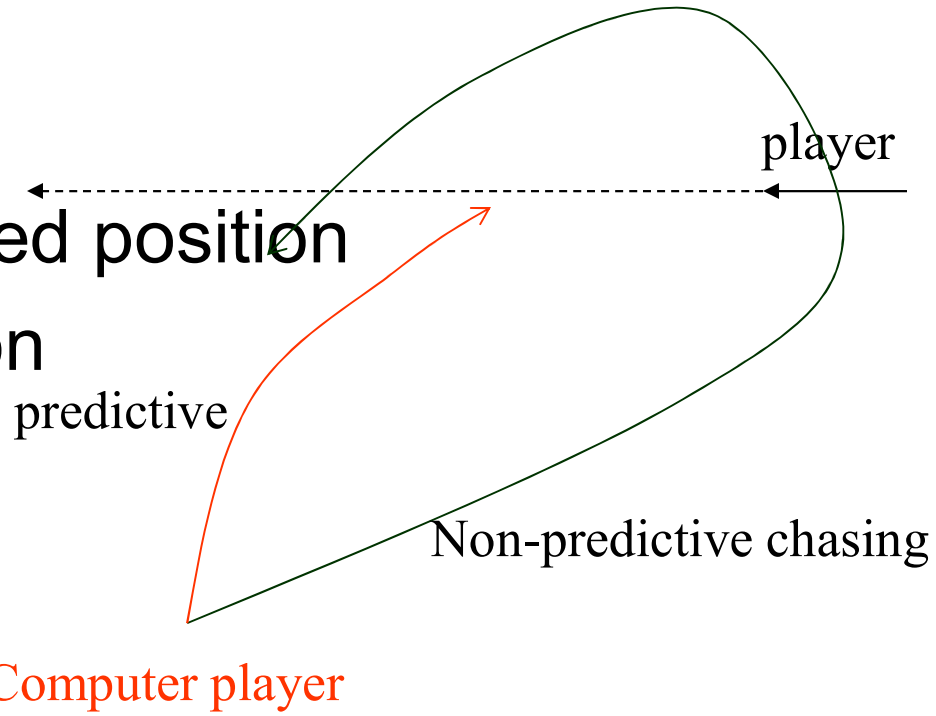
- usually implemented as working in 2D
- Keep aligned with the target and advance towards it

```
self.ideal_yaw = enemy_yaw;
ChangeYaw();// rotate to face enemy
self.yaw_speed = 50;
walkmove(current_yaw, 50);
```


Chasing

- Predictive chasing can make the enemy more intelligent

1. Calculating a projected position
2. Aiming at that position
3. Advancing



Memory

- Storing AI data(*knowledge representation*) is complex
- Simpler on individual level
- Often end up with case-by-case solutions

Analysis/reasoning Core

- *Finite State Machines*(FSMs) and *Rule Systems*(RS) are usually used
- Making a decision can be fast or slow, which depends on the number of alternatives and sensory data

Action/Output System

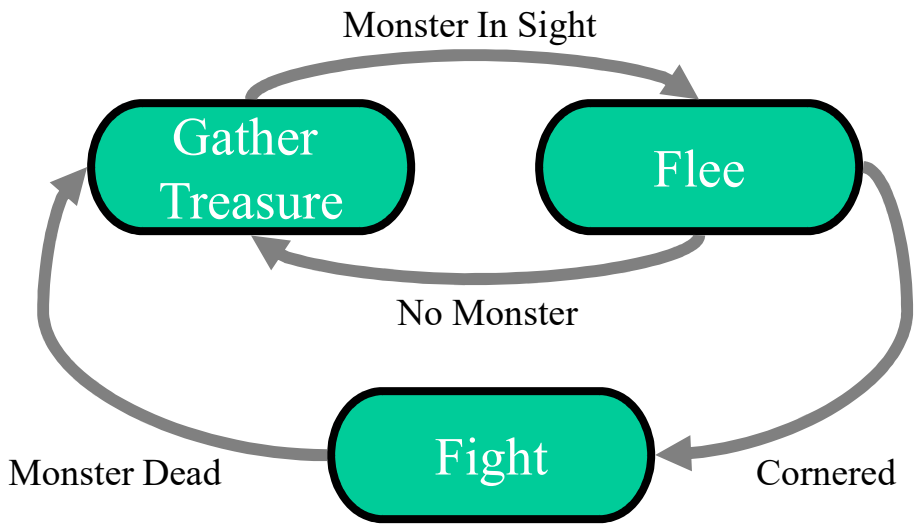
- Collision detection & response requires geometrical tests
- Primitives tests involved – point, triangle, sphere, object ..
- Depends on the abstraction of the programmer & speed requirement

Finite State Machines(FSM)

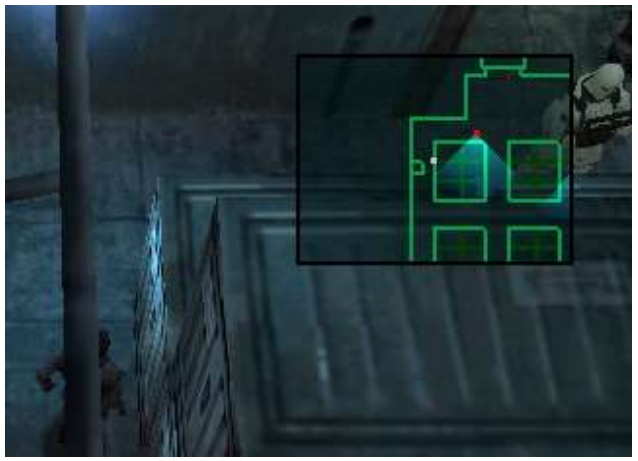
- Also called *Finite State Automata*
- Consists of
  1. a set of states that represent the scenarios or configuration
  2. A set of transitions that are conditions that connect two states in a directed way

Finite State Machines(FSM)

- Character AI modeled as a sequence of mental states
- World events can force a change in state.
- Intuitive and easy to code

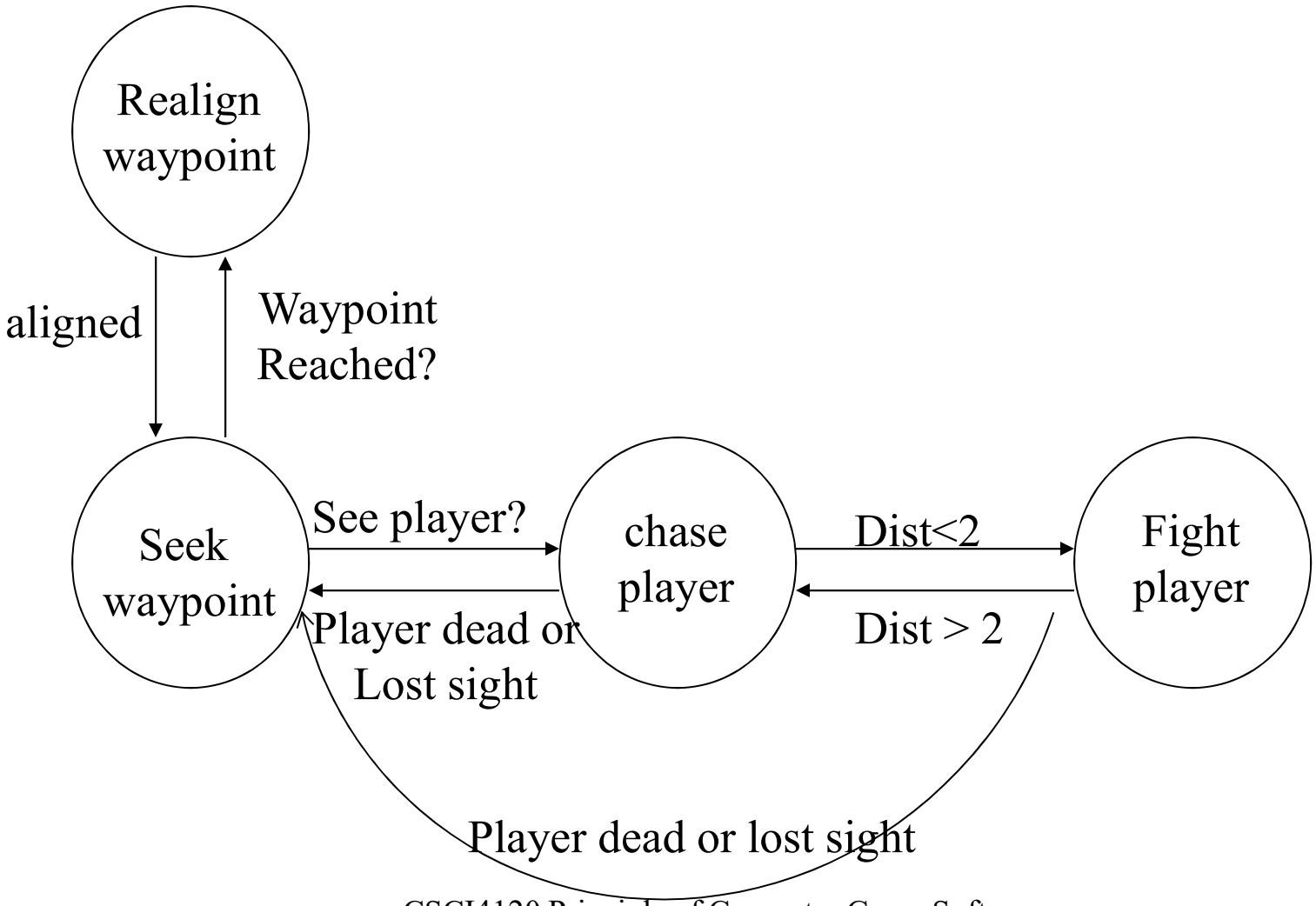


Finite State Machine



- Example of a guard
- First draft the spec. list:
  1. Outdoor area with no obstacle
  2. The guard has a predefined waypoints he patrols in a cyclical way
  3. Activates when you get inside his viewing cone
  4. If he sees you, he will chase you
  5. He carries a sword and if in close contact, he will stop and hit you with the sword

Graphical layout



Map to source code

- Simple
- Reserving 0 for initial state & positive integers for the others
- Code can be written in this way

```
case [NAME OF STATE]:
    [DEFAULT ACTIONS]
        [CONDITION EVALUATION]
    if (TRANSITION)
      state=destination state
    (..)
    break;
```


Map to source code

```
#define       SEEK_WAYPOINT 0 // initial state
#define       ROTATE_WAYPOINT      1 // state
int state;
switch (state){
   case SEEK_WAYPOINT:
         // code for this specific state
            break;
    :
    case ROTATE_WAYPOINT:
         // code for this specific state
               break;
    }
```


Nondeterministics Automata

- Classic FSMs are *deterministics* – behavior of the AI entity is totally predictable
- Introduce limited degree of randomness
- A state might have several transitions, which can activate or not in *controlled* random manner
- e.g. the guard may stop for a cigarette when reach a waypoint

Nondeterministics Automata

- Useful in games where many AI entities are using the same state machine

Summary on FSM

- Advantages
  - Intuitive, easy to code
  - Can model unit AI in complex game world

Drawback

- Requires much duplication of similar code on various states due to behavior
- No memory on previous states e.g. after transit to attack states and completed, can't return to its original state such as talking etc.
- Can't scale up well
- Has difficulty in concurrency – deadlock easily happened on managing external resources

Rule Systems

- FSMs are well suited for modeling behavior where
  - only a few outcomes are possible
  - actions are sequential in nature
- For behavior that operating by priority, rule system will be better

Rule System

- Consists of set of rules

  Condition -> Action
- A rule closer to top will have precedence over those near bottom
- Suitable to model behavior that is based on *guidelines*

Implementation(Symbolic)

- Using a *scripting language* which symbolically represents the rule
- Rules are written according to spec., parsed from external file and executed in real time
- Used in real time strategy to control the tactical level of game play

Age of Empire

Implementation(Symbolic)

- Example in Age of Empires
- **defrule**

  (resources-found wood)
  (building-type-count-total lumber-camp < 5)
  (Dropsite-min-distance wood > 5)
  (can-build lumber-camp)
  $\Rightarrow$ **(build lumber-camp)**

- Usually tactical reasoning requires a lot of rules, e.g. 50
- Rule evaluation need to be optimized

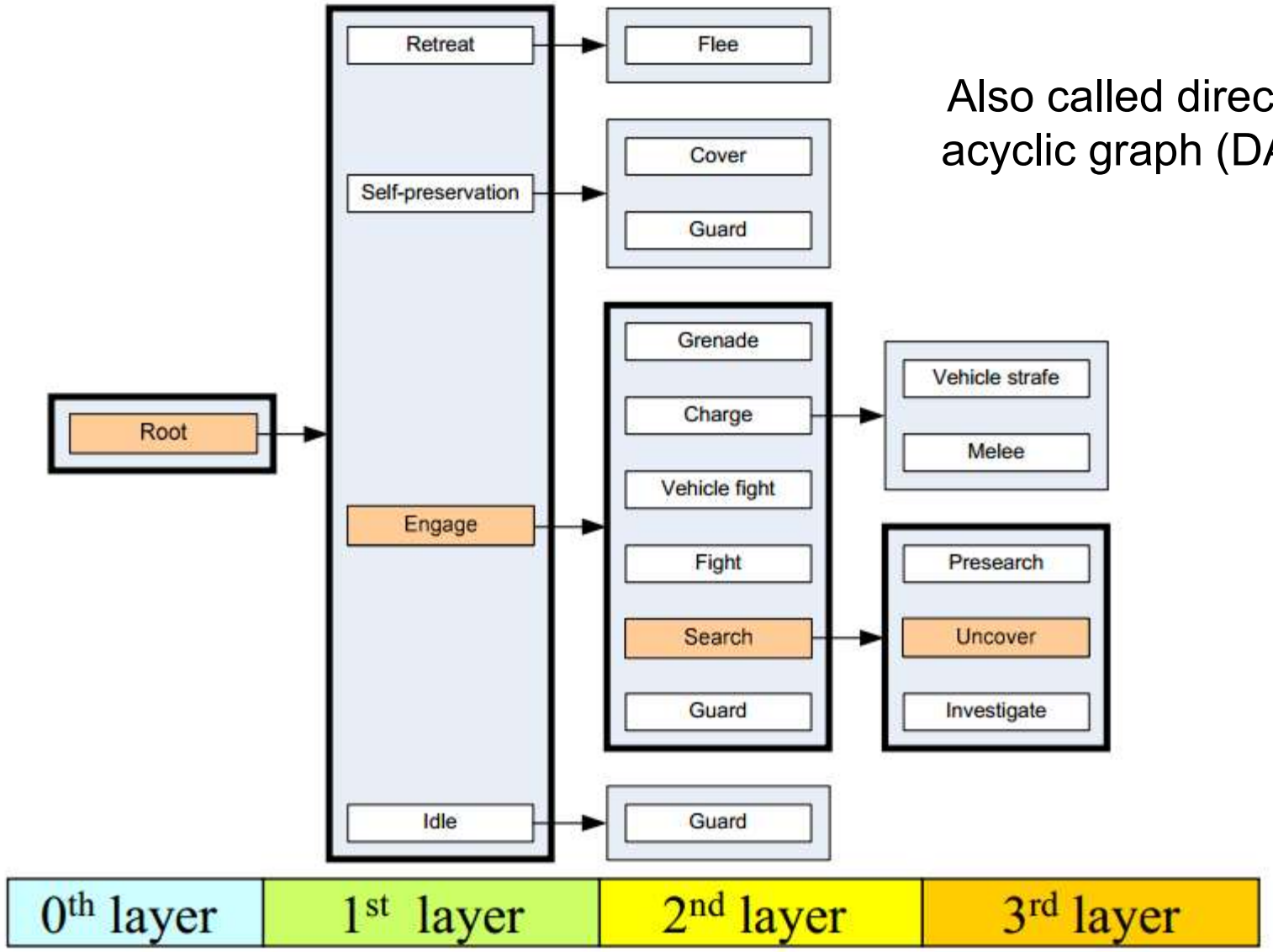
Problem of RS

- Variable duration AI tests - rules are explored sequentially until one of them is true
- If no rule is valid, a significant processing time might need to reach the final rule
  - Can be improved - limiting the evaluation effort after a first iteration by collecting/caching facts already known eg Rete algorithm
- Problems in handling complex game worlds in 3D

Hierarchical FSM

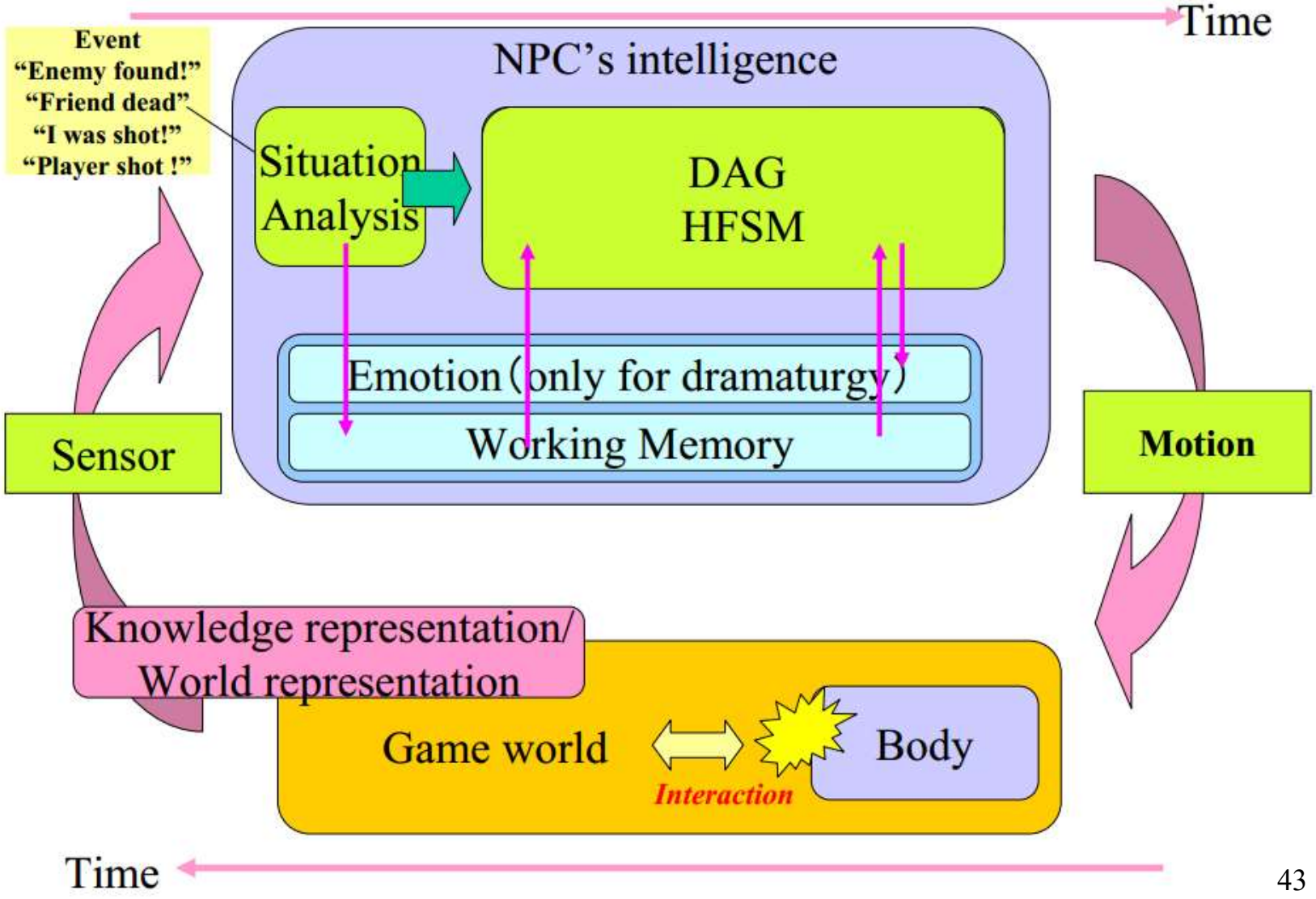
- Classic FSM is memory less – causing inconsistent behavior for NPCs in game
- Added is the problem in scaling up and maintenance
- Need technology that can cope with much complicated game world nowadays

Hierarchical FSM

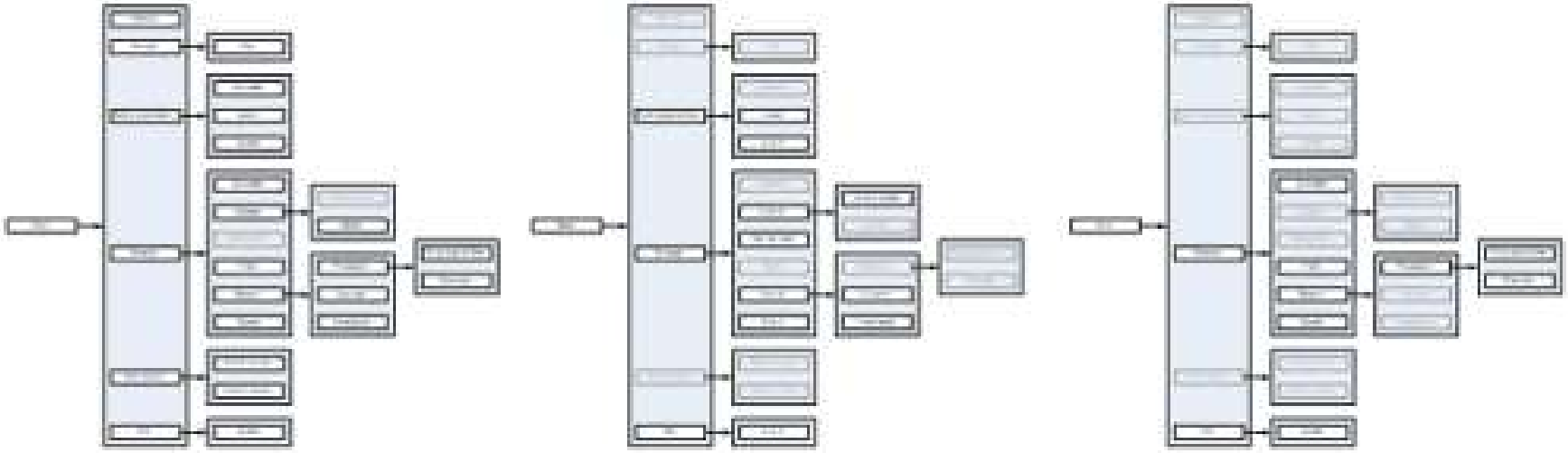


Also called directed acyclic graph (DAG)

Halo2 NPC Architecture



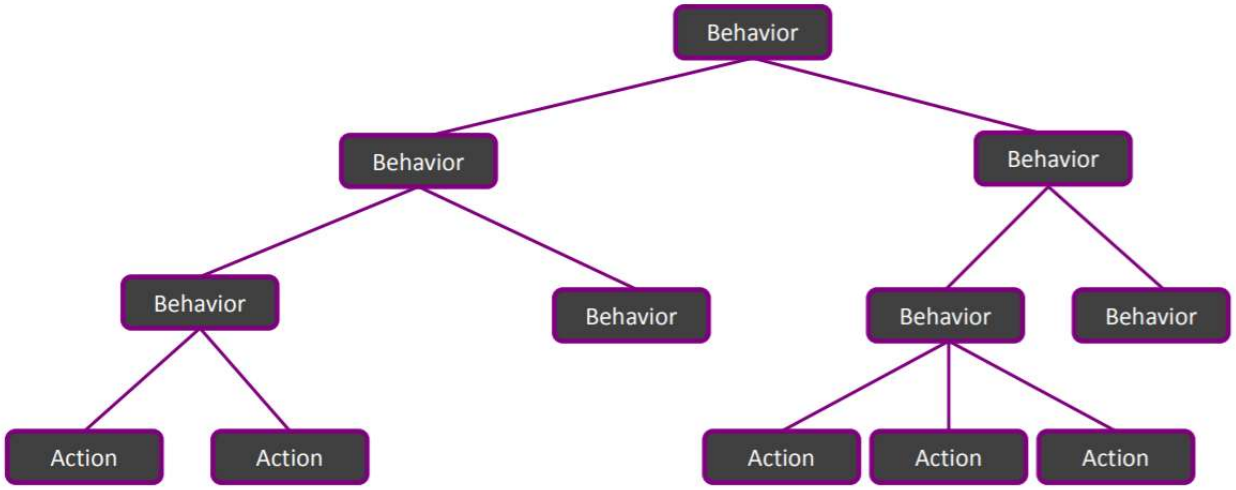
Halo2 NPC Architecture



- When scaled up for different character class, complexity up also
- This rapidly growing complexity is a problem for all game AI developers

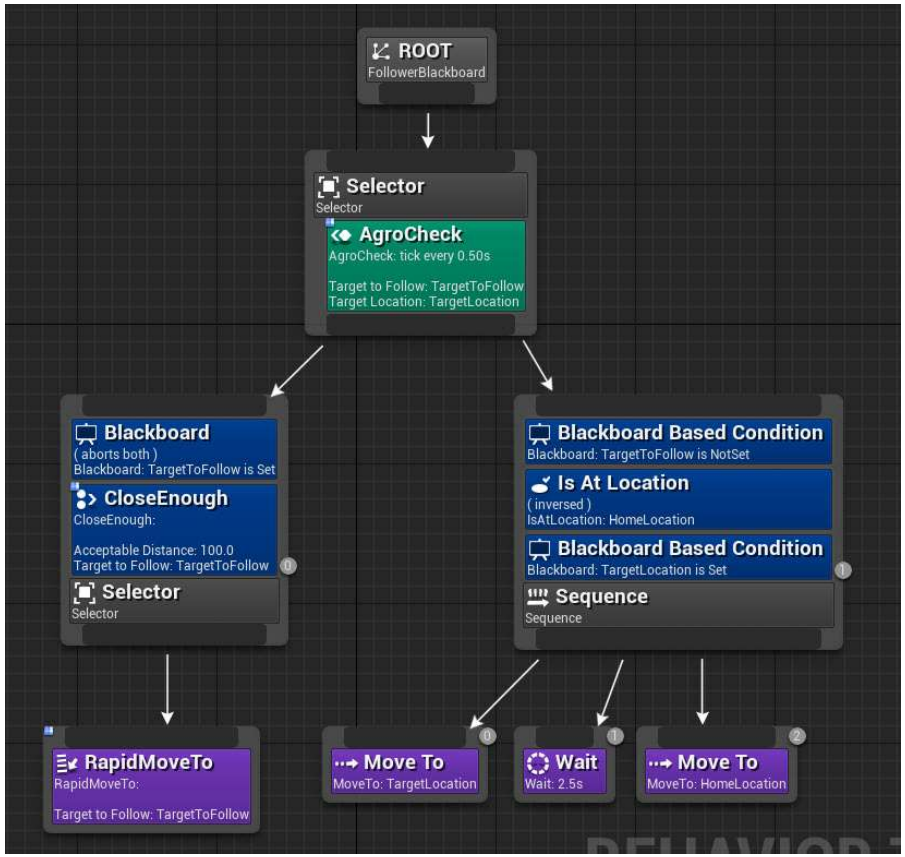
Behavior Tree

- Describe switching between a finite set of tasks
- Create very complex tasks composed of simple tasks
- Similar to HFSM with difference building block is now a task



Behavior Tree

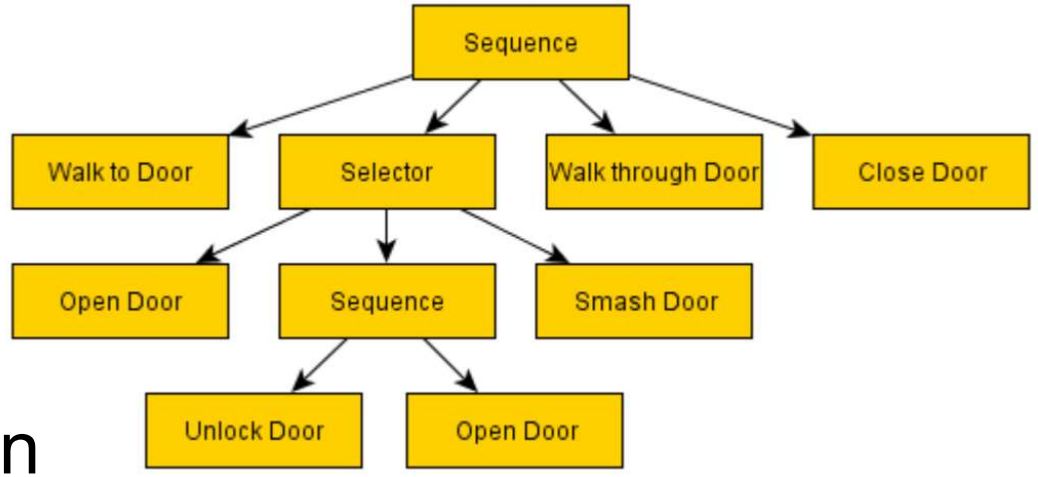
- Node types
- Root
- Priority (Selector in Unreal)
  - Child nodes are evaluated in order until one validates
- Sequential (Sequence)
  - First child validated & execute
  - Next one is validated
- Decorator
- Operator, only 1 child



BT in Unreal

Behavior Tree

- Running of BT
- Execution starts from root, sending ticks to child
- A tick enable execution of a child
- Child will return
  - Running : not finished
  - Success : achieved goal
  - Failure : otherwise



Depth-first traversal

Behavior Tree

- Good

1. Separate algorithm from behaviour
2. Support node/tree reuse
3. Support dynamic behaviors (attaching BT to particular actor in level at run time)

- Issues

1. Cost of evaluating large BT can be prohibitive
2. Only better organizes behaviors, does not provide a model for decision making

Planning & Problem Solving

- Some problems need analysis, reasoning, thinking that are not easily handled by *FSM* or *RS*:

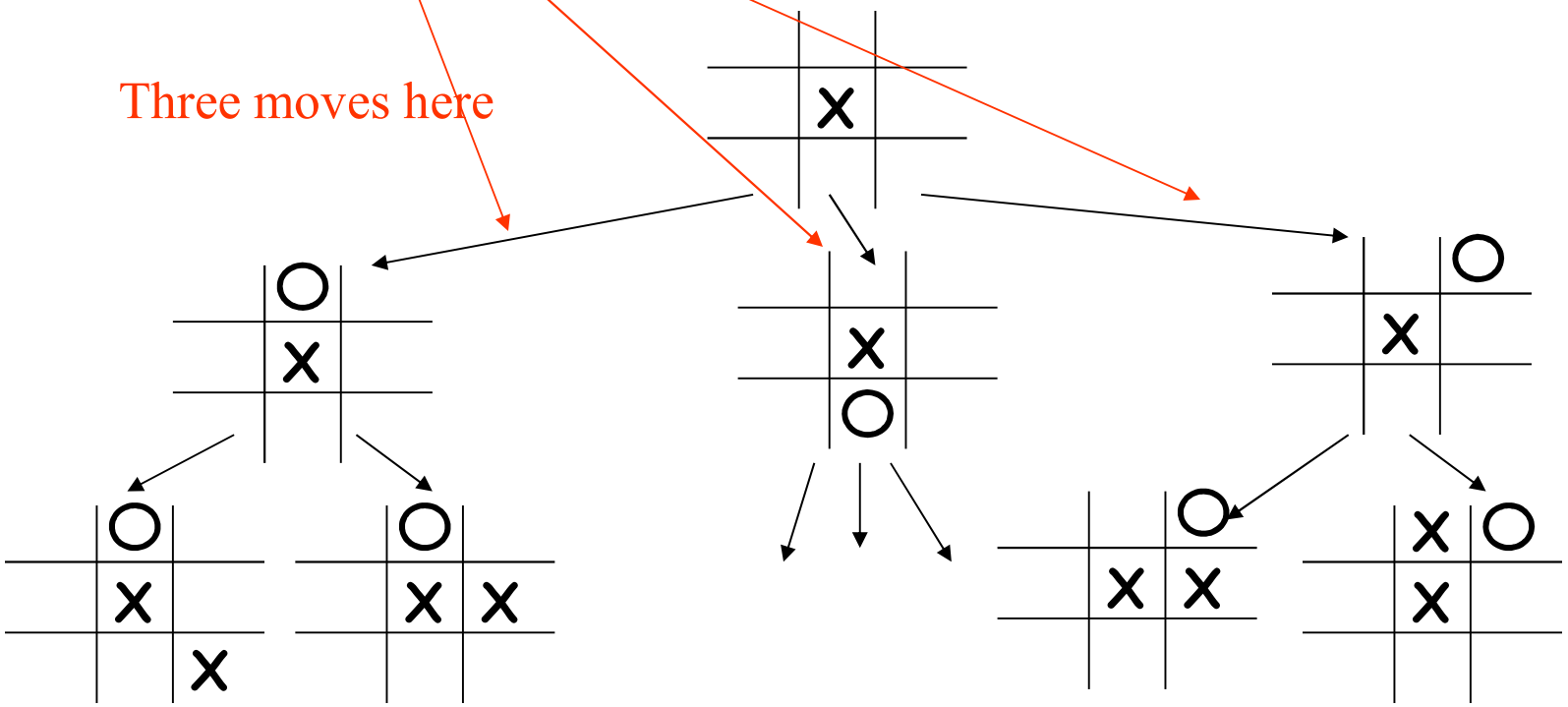
1. Puzzle solving
2. Chess playing
3. Select best route to attack N ground targets in flight sim.
4. Trace a path from A to B with obstacles in between

Planning & Problem Solving

- These problems, say chess playing, can be modeled as consists of a large number of states
- Candidate transitions are evaluated for suitability for reaching of goal – *state-space search* problem
- Represents the game as a tree with possible configurations as nodes in the tree – *game tree*

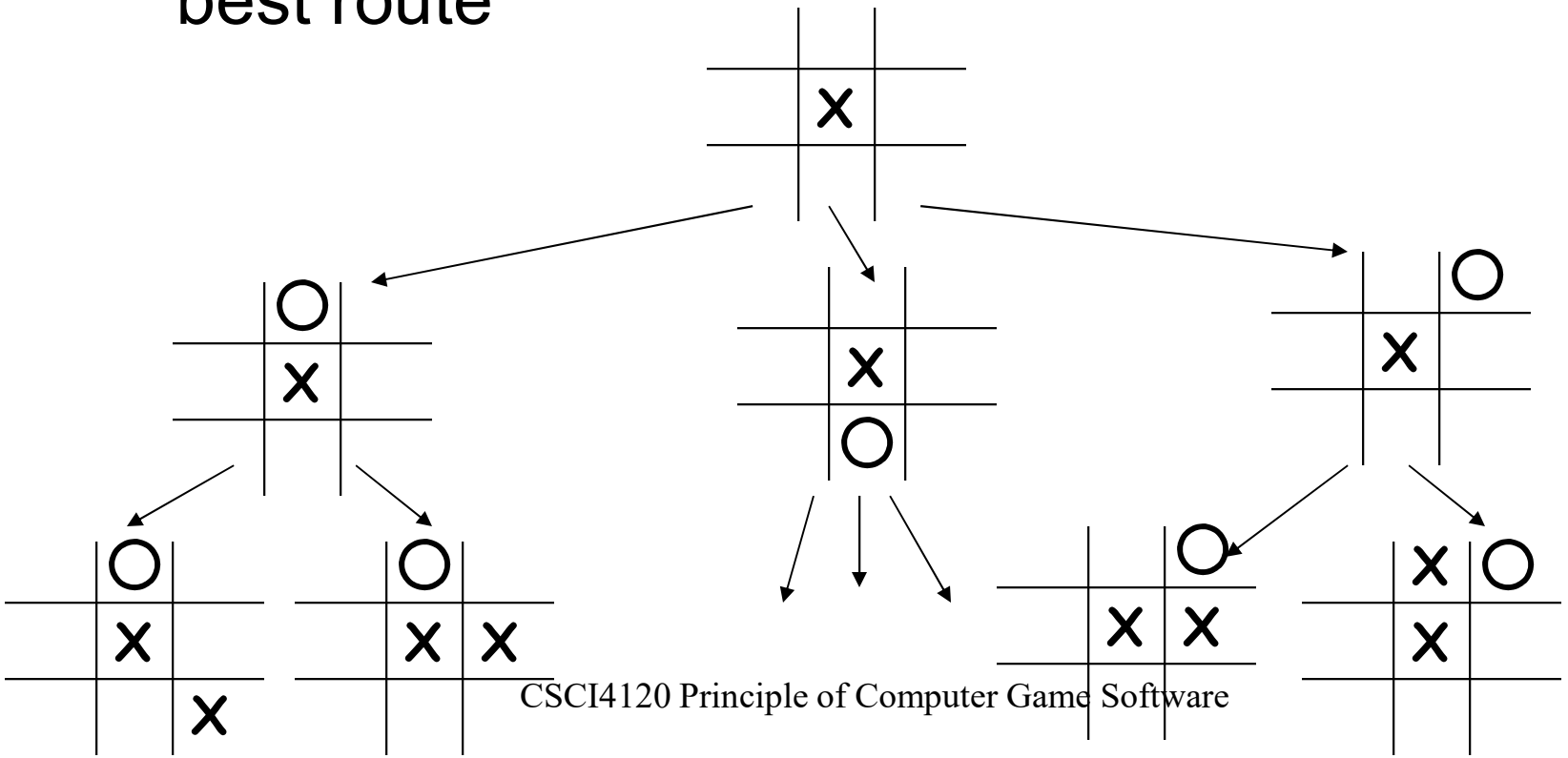
Game Tree

- Moves is modeled as branches between states
- Need an evaluation function – quantify how good a particular game position(state) is



Planning & Problem Solving

- Computer player works by considering how far it can into the future from current position
- Problems: Given initial & target state, find the best route



State-Search

- *Depth-first search* – recurs further until no more move. If final state is not solution, start all over again
- Can take very long time to compute a solution
- Sometimes exhaustive search not possible – consider chess playing
- Use heuristic based search – *A* search* (path finding, later)

Biological-Inspired AI

- Try to simulate the cognitive process
- *DNA* codes varied only a little among people, but yields different individual
- Each generation create a new generation whose DNA consists of a combination of the parents DNA plus a minor degree of *mutations(changes)*
- Nature perform task of filtering i.e. *fittest will survive*

Evolutionary Computation

- Test the generated individual against their ecosystem and measure their performance
  1. **Generate the first N individuals with random DNA**
  2. **Perform fitness testing**
  3. **Select top percent**
  4. **Generate N descendants that are**
     1. **Linear combination of parents DNA**
     2. **mutations**
  5. **Go to (2) until reach desired number of generations**

Fitness testing

- *Automatic* or *supervised*
- Automatic : defining function that can be computed automatically and apply to all individuals
- Supervised: user select the top percent

Evolutionary Computing

- Hard to define automatic fitness test in some instances e.g. evolved species converge
- Supervised selection will be needed – user perform the *Darwinian* selection (進化論)
- System must evolve in relatively small number of iterations
- Time required to evaluate & select from one iterations must be brief

Evolutionary Computing

- *Mutations* are introduced to the breeding process – add some random variations
- A phenomenon known as *local minimum* in a global minimum problem – *steepest descent* might make the solution stuck in local minimum
- random perturbation can bring the algorithm out of the local minimum, and thus seek for global minimum

Learning

- Learn from imitation, mistakes, experience & trial and error
- AI system learns at the same time with the player => AI allows the opponent to adapt as the player learn skills & strategies

- Example: procedural narrative

Learning

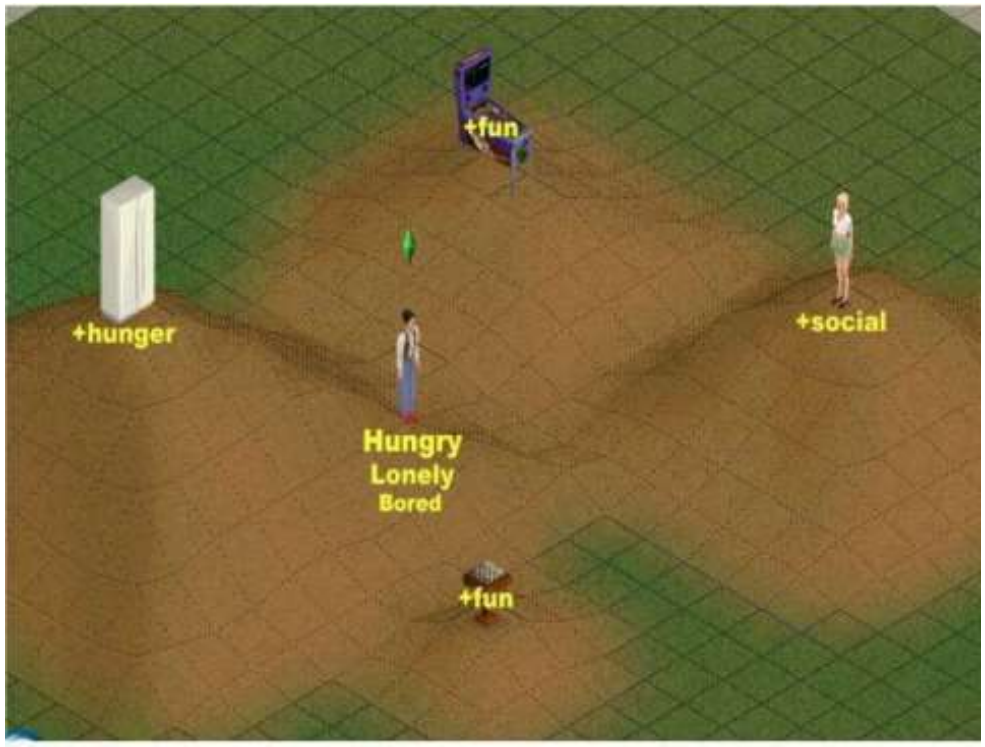
- Forza used learning neural network to control NPC drivers
- the system watches you play and imitates your driving style

- Recent release use cloud services to download AI racers based on other human players
- Opponents then mimic players around the globe

Forza

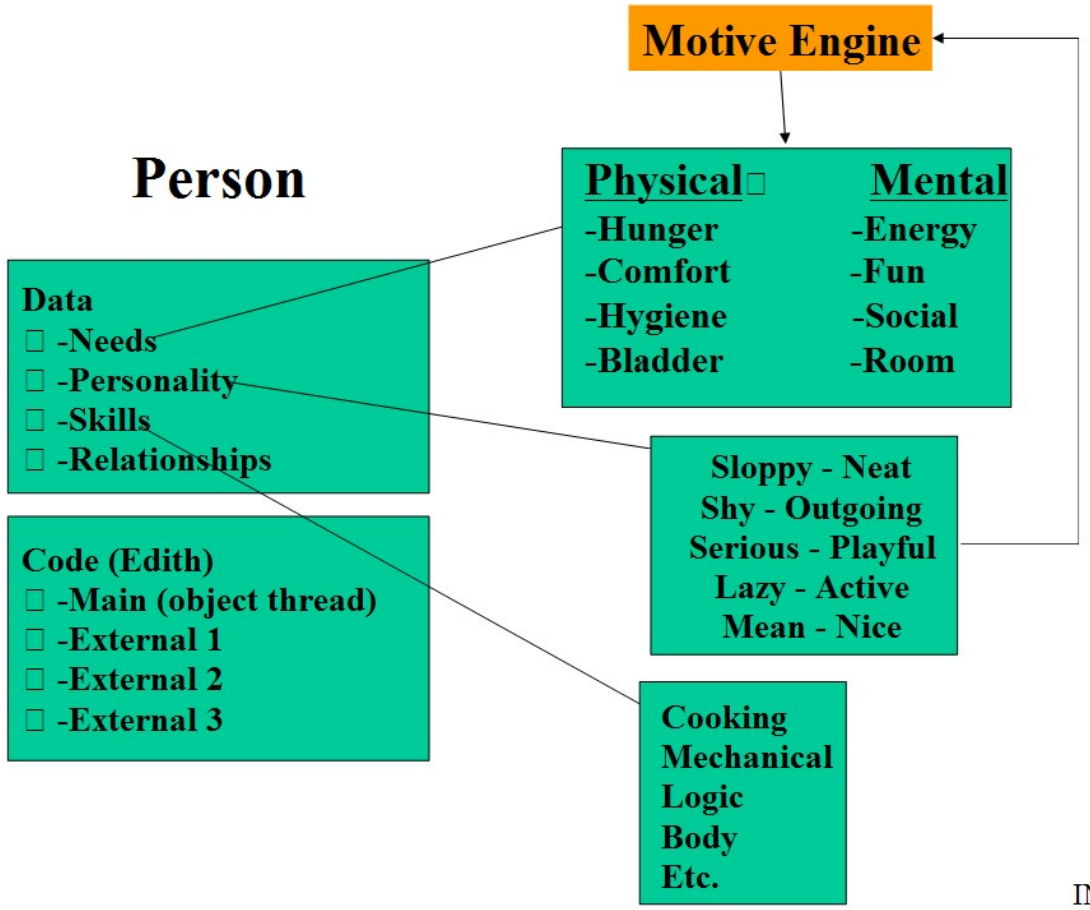
Autonomous AI

- Battle AI – AI in action games is easier to develop than AI in real world (daily AI)
- Daily AI is having much variety



Autonomous AI

- Data structure for NPC in Sims

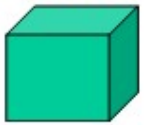


Find Best Actions

Hunger +20
Comfort -12
Hygiene -30
Bladder -75
Energy +80
Fun +40
Social +10
Room -60

Mood +18

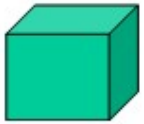
Toilet



Mood +26

-Urinate (+40 Bladder)
-Clean (+30 Room)
-Unclog (+40 Room)□□

Bathtub



Mood +20

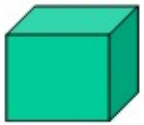
-Take Bath(+40 Hygiene)
□□□ (+30 Comfort)
-Clean □□□ (+20 Room)
□□

Find Best Actions

Hunger +20
Comfort -12
Hygiene -30
Bladder -75
Energy +80
Fun +40
Social +10
Room -60

Mood +18

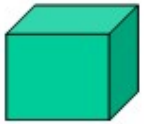
Toilet



Mood +26

-Urinate (+40 Bladder)
-Clean (+30 Room)
-Unclog (+40 Room)□□

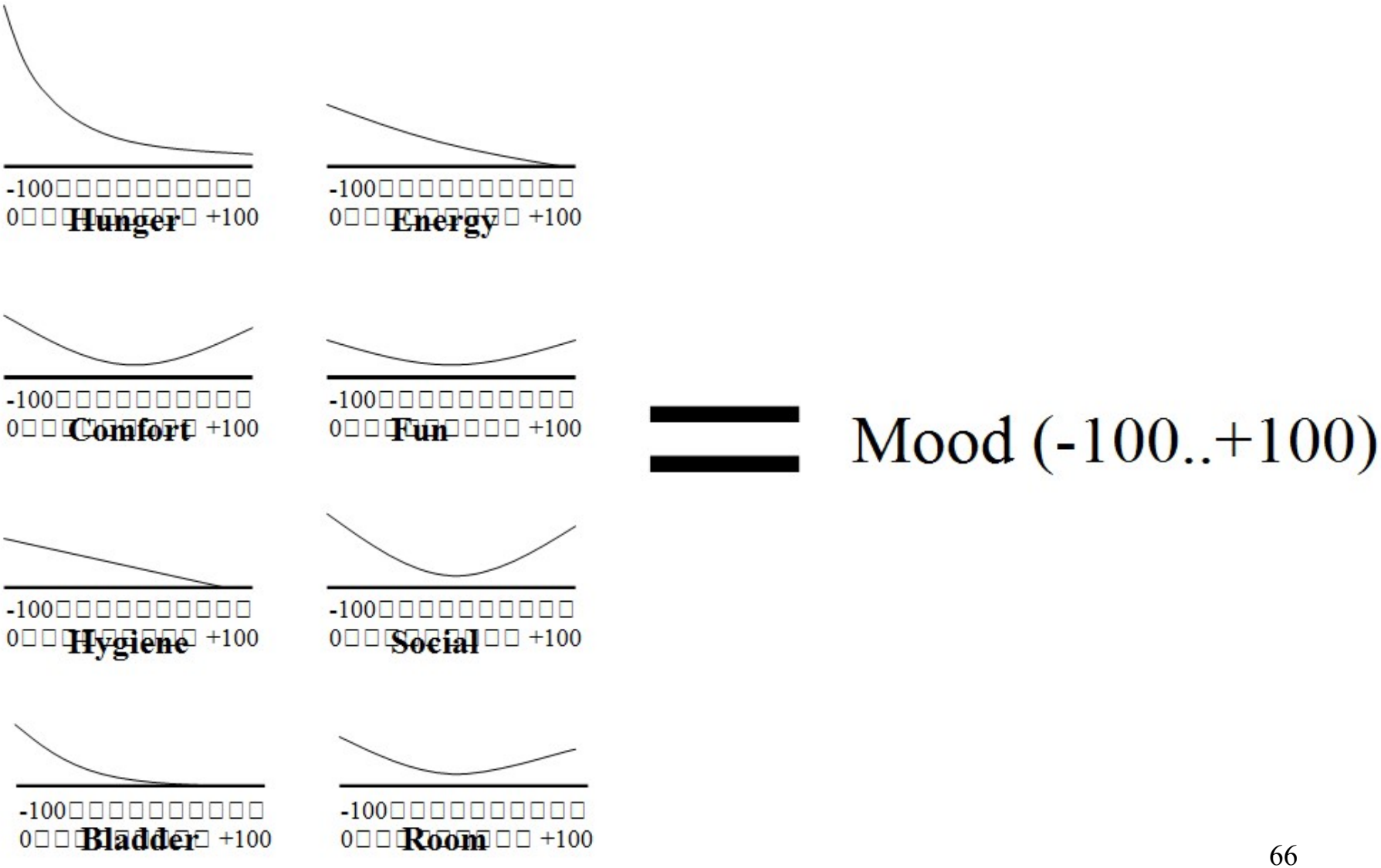
Bathtub



Mood +20

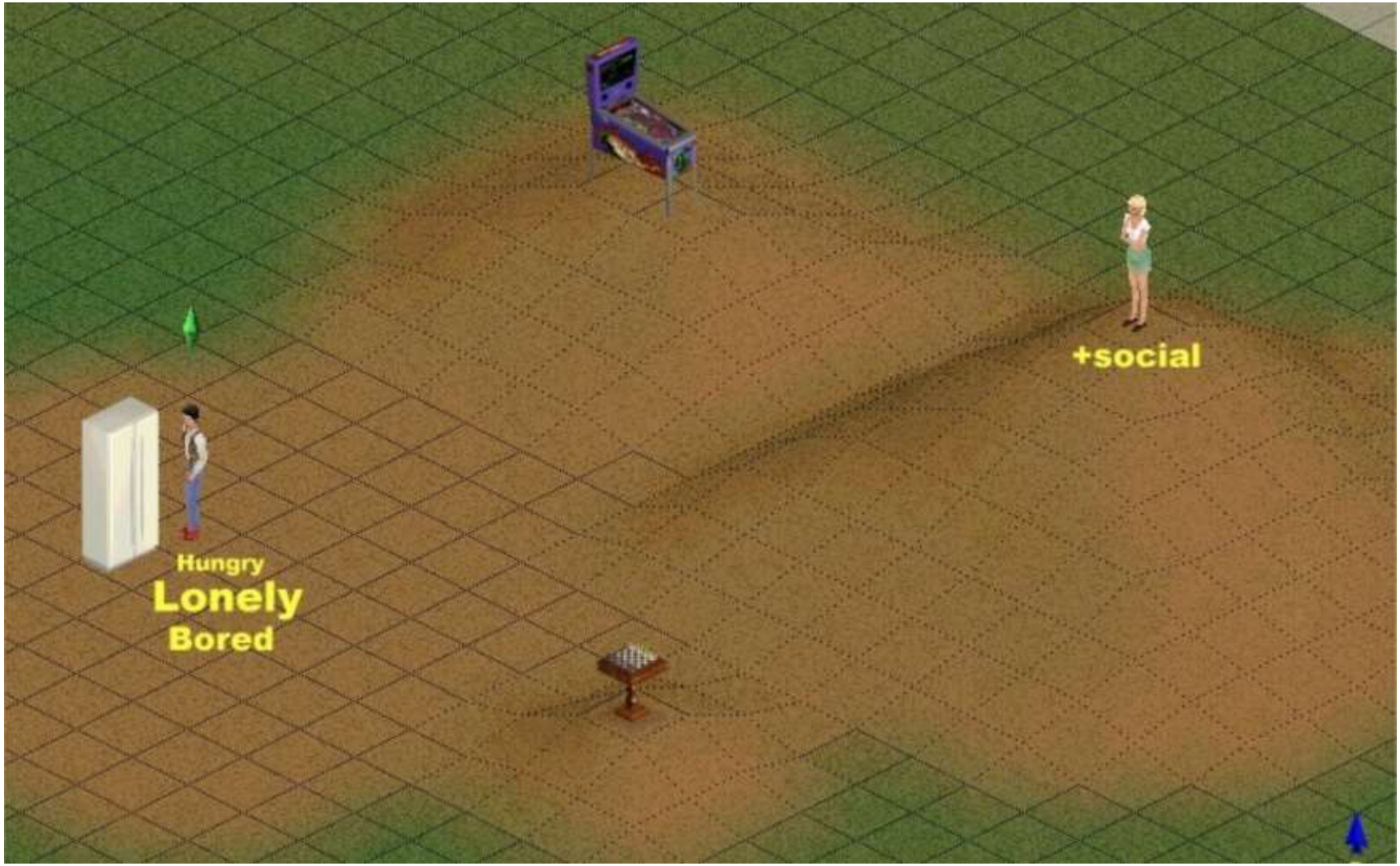
-Take Bath(+40 Hygiene)
□□□ (+30 Comfort)
-Clean □□□ (+20 Room)
□□

Happiness Weights

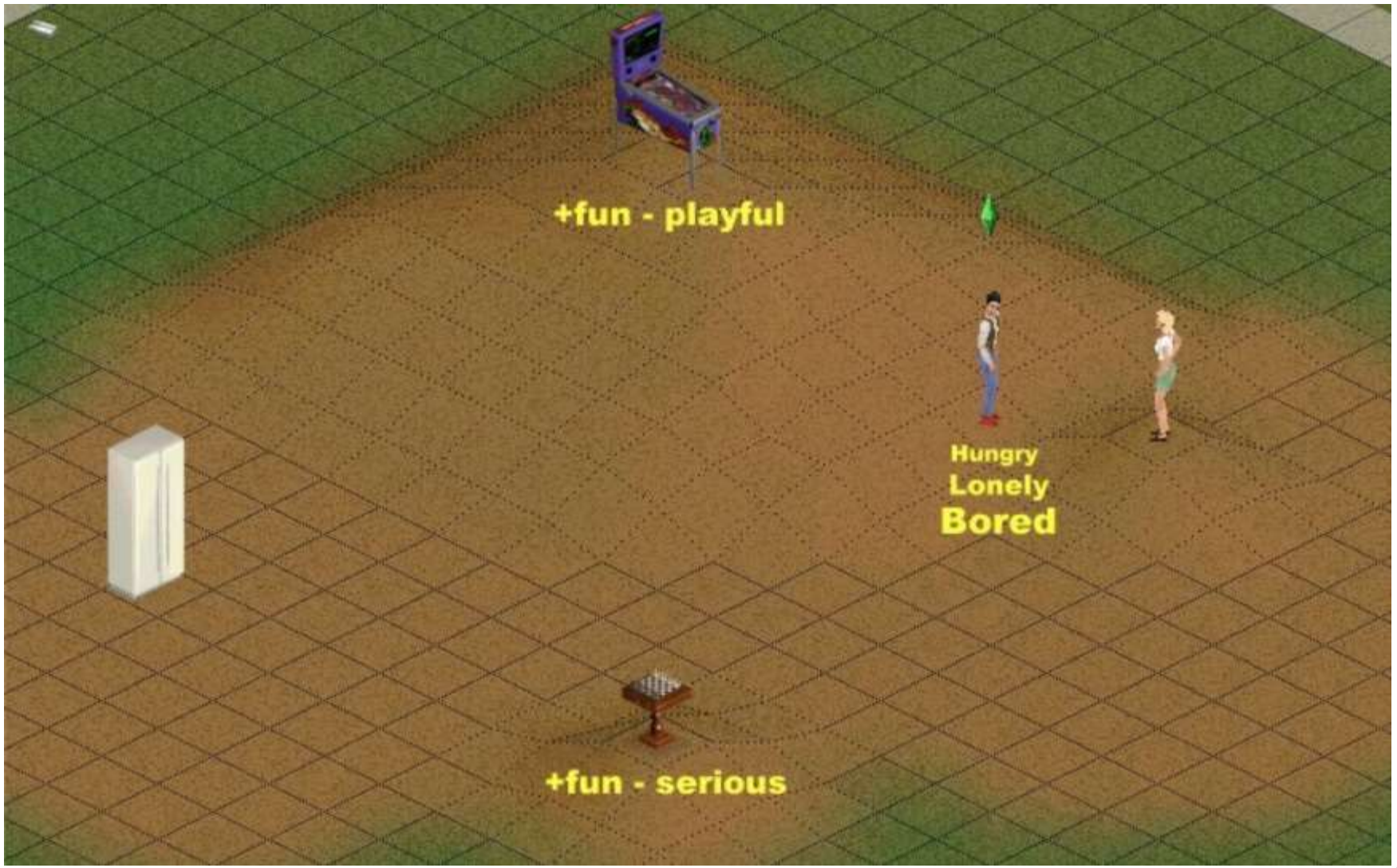


Maximize Happiness

Maximize Happiness



Maximize Happiness



Machine Learning in Game AI

- Procedural narrative - Introduced in “left for dead“(2008)
- introduce storytelling into multiplayer game
- AI will try to build up a model of the players captures the skills, weaknesses, preferences, and other characteristics of that player
- e.g. moving together as a group or splitting up? How is the response to a particular creatures?

left for dead“(2008)

Procedural Narrative

- **Pacing and Events**
  - AI can query the player model to determine how best to adapt its behavior to that particular player
  - e.g. if they've been particularly challenged by one kind of creature then decisions can be made about how to use that creature in subsequent encounters.

- more of a story-telling device than, say, a simple difficulty mechanism.

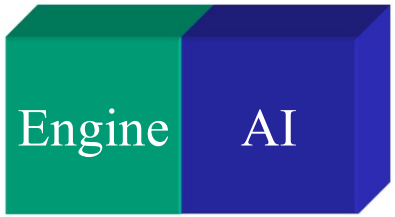
Important Dates

- Final Exam
- Date : Dec 7, 2021
- Time : 7:00 – 9:00pm
- Venue: WMY 505 (capacity 90)

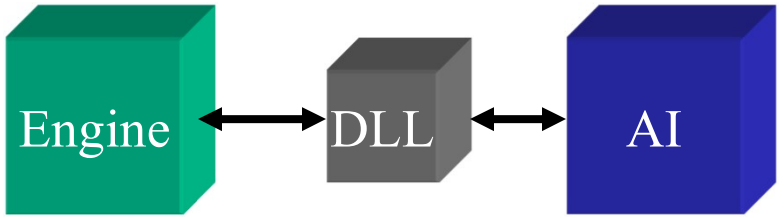
- Project Phase 2 Submission
- Date : Dec 24 (Friday)
- Grade submission on Dec 28

Game Engine Interfacing

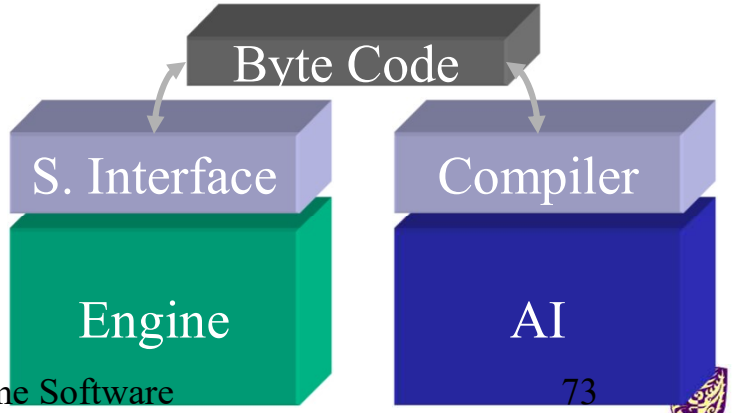
- Simple hard coded approach
  - Allows arbitrary parameterization
  - Requires full recompile



- Function pointers
  - Pointers are stored in a singleton or global
  - Implementation in DLL
    - Allows for pluggable AI (QuakeBot).



- Data Driven
  - An interface must provide glue from engine to script engine (UDK, Reality factory etc).

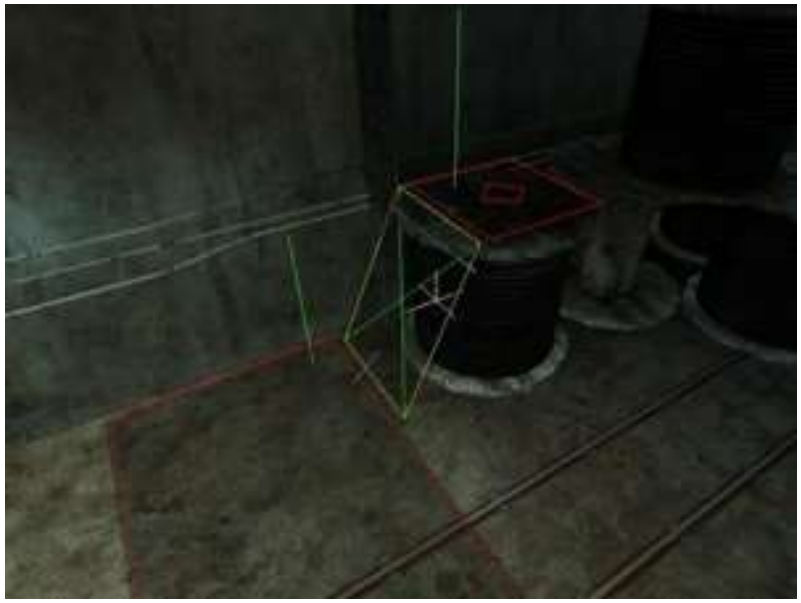


Tactics

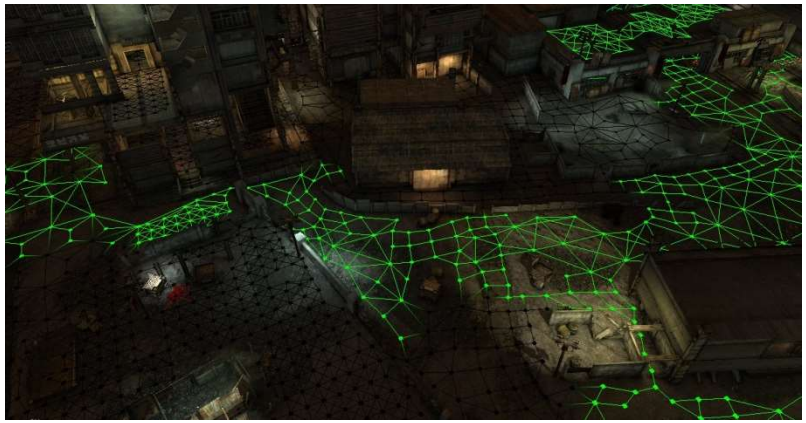
- Typical in game AI achieved through level designer placed hints and scripts
- Scripts
  - define how AI responds to specific situations e.g. FSM
- Level designer placed hints
  - mark cover and ambush spots
  - Static, in the form of either navigation mesh or way points

Tactics

- Navigation mesh
  - represents the "walkable areas" of a map
  - Enable AI entities to navigate within the level
  -
- Need level designer further editing to enhance AI performance



Navigation mesh,
Area with Green X marks the jump
Area to tell bot to jump up & reach higher area



Tactics

- Way points
  - More precise in designating locations
- make players believed their opponents are intelligent
- Drawback
  - only valid for prescribed actions
  - Cannot cope with changing environment e.g. destructible scene & moving vehicles

Positions in Halo 2

Tactics

- A tactic consists of
  1. Initial state
  2. A goal state
  3. Plan to move from one state to another
- In game correspondence:
  - move a unit from a point in map to destination
  - to use a weapon when current position in technology tree is not possible

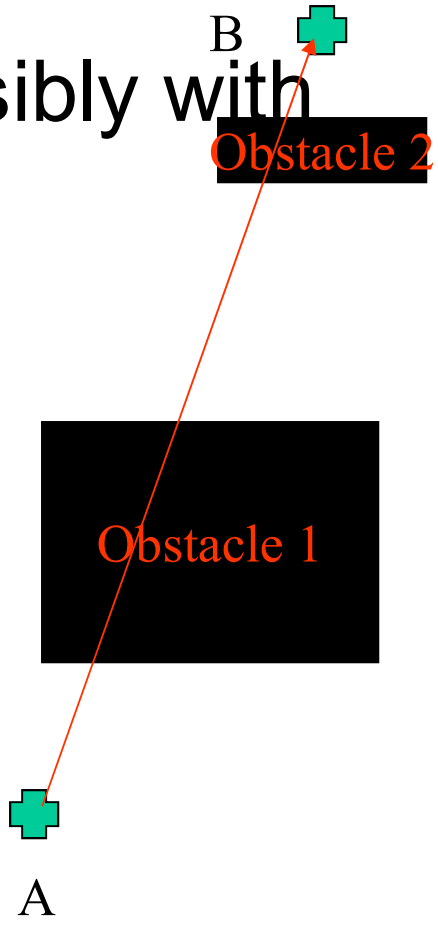
Machine Tactics

- Classified into the following problems

1. Move intelligently in a scenario(path finding)
2. Analyze a scenario(geometry, enemies etc)
3. Create a plan by selecting right actions to achieve a goal

Path Finding

- Problem:
  Move from *Pt. A* to *Pt. B*, possibly with obstacles
- Local & global algorithms



Path Finding

- *Local*: Analyze the surrounding of current position only (action games)

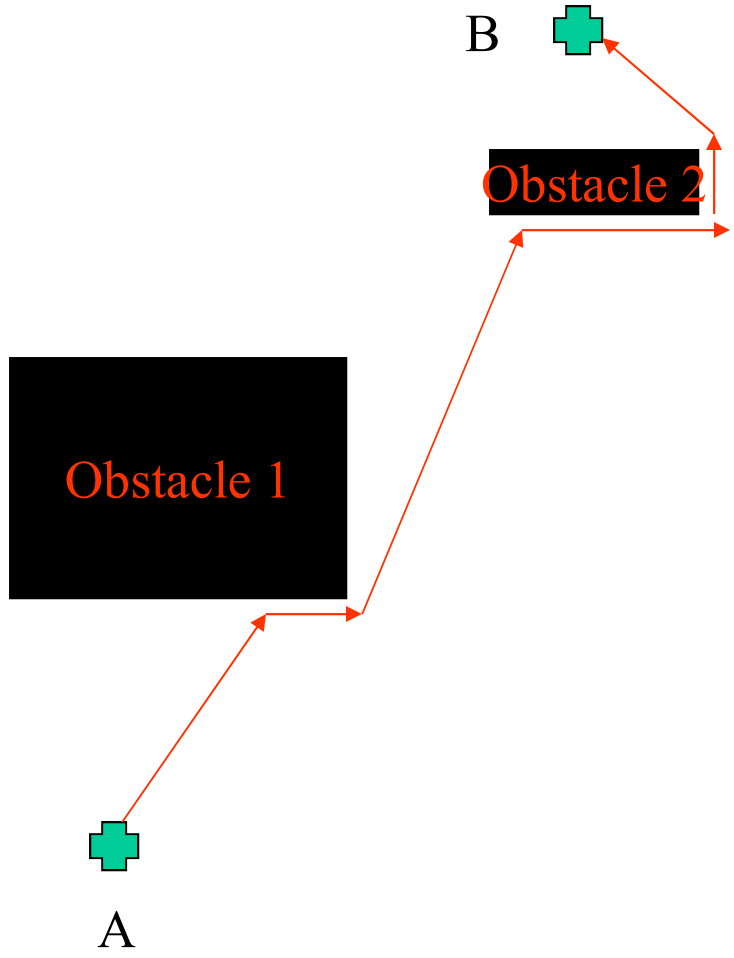
- Check previous discussions on Quake implementation

Path Finding

- Global: Analyze the area as a whole and trace the best path(strategy games)

Crash & Turn

- Strategy
  - move in the straight line connecting two pts., if reaching an obstacle, trace along the boundary until have open line of sight of destination

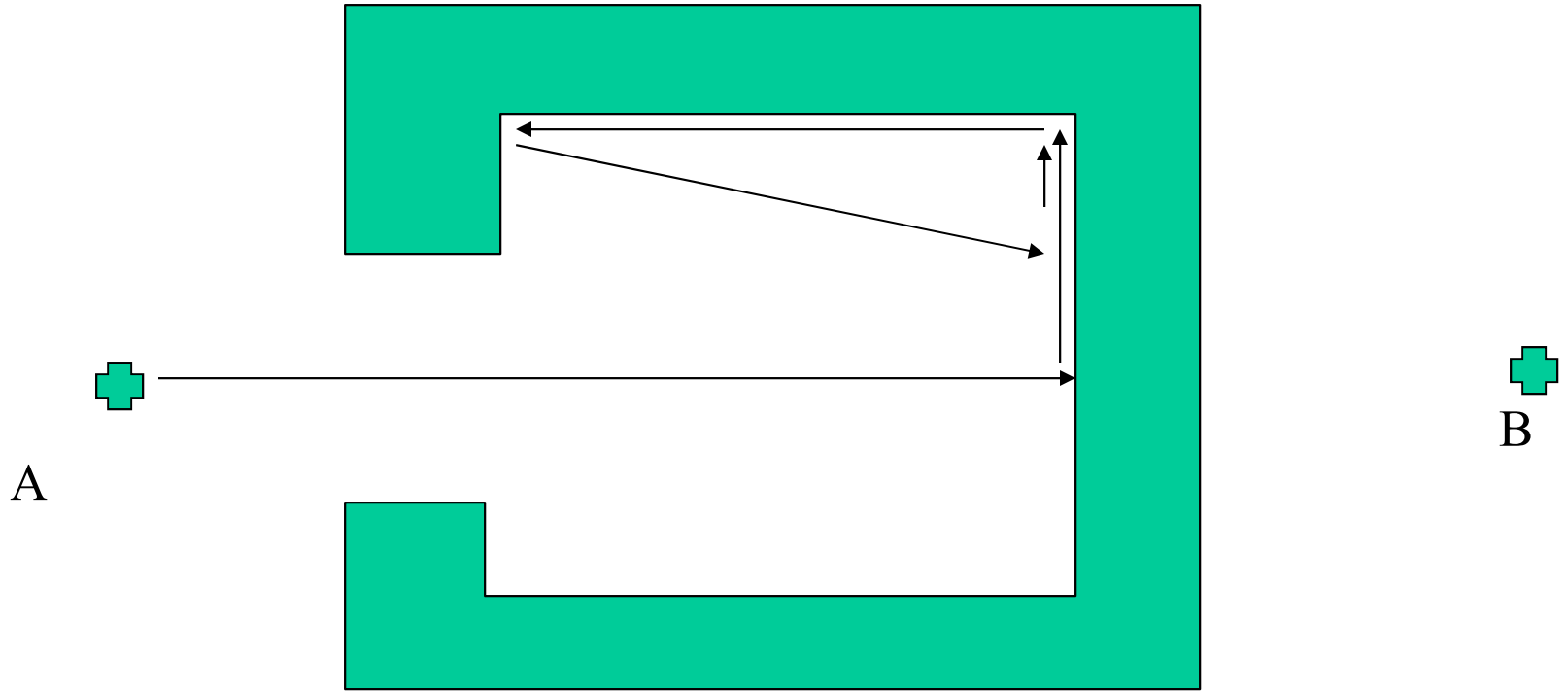


Crash & Turn

```
While (not at destination)
   if can advance in straight line
         advance
   else
         select left/right direction
         advance with left/right touching obstacle
```


Crash & Turn

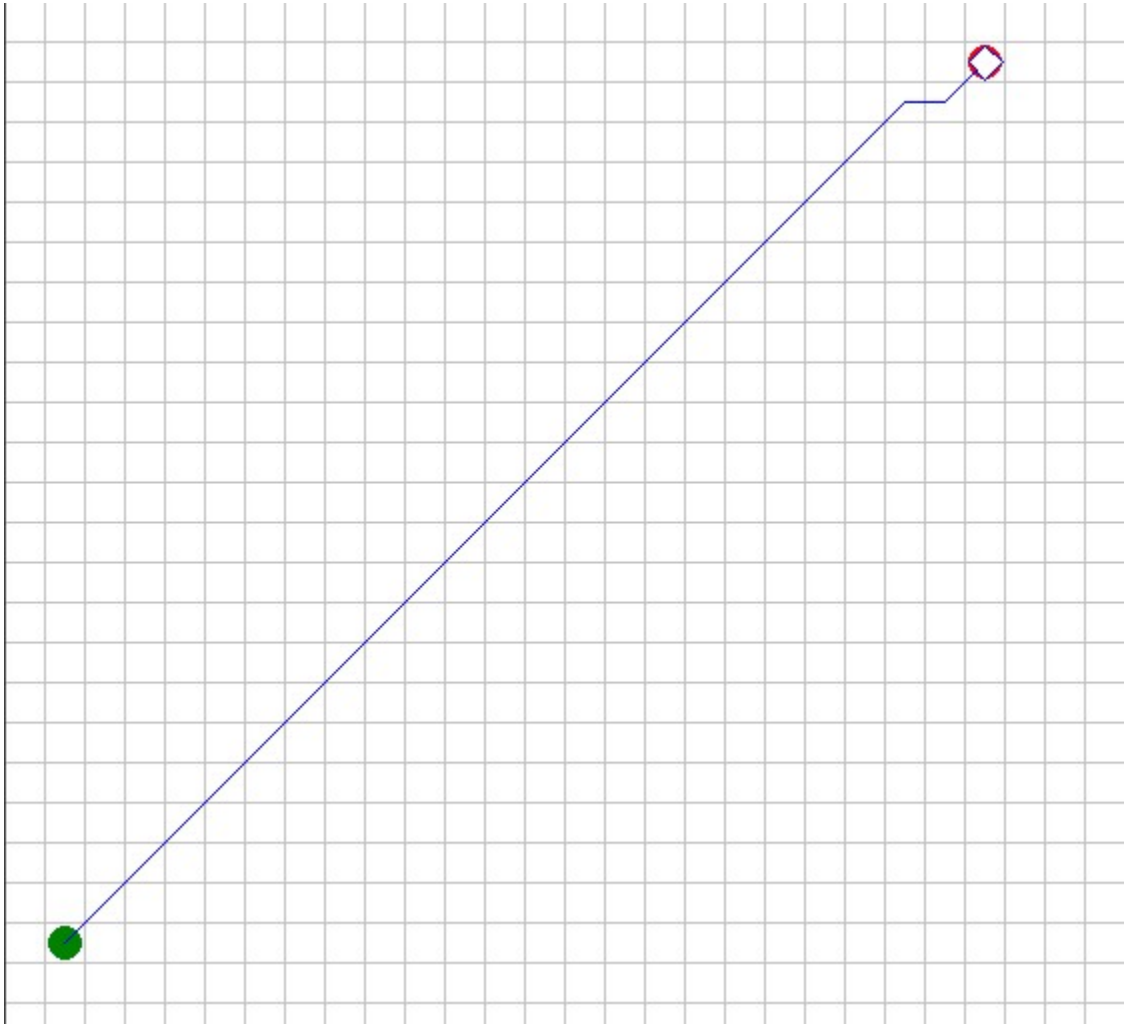
- **Only suitable for convex obstacles**



More robust approaches

- Plan the entire route before start the first move
  - Divide the space into a number of *tiles*
  - Movement between tiles is modeled as *state transition*
  - Find a transit path from initial state to goal state
- Note: A discretized approach i.e. difficult to use in continuous world such as 3D FPS

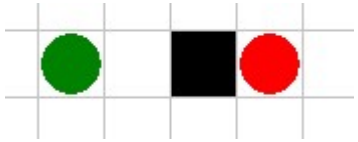
Tiled World



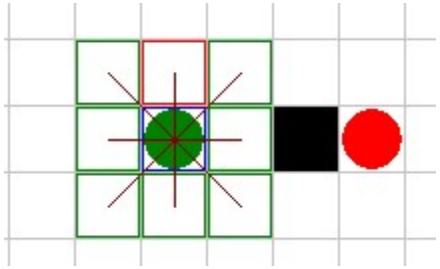
Breadth-first Search

From initial state
while (not at goal)
    advance a tile along all possible directions
    Record all paths and check whether arrived at goal state

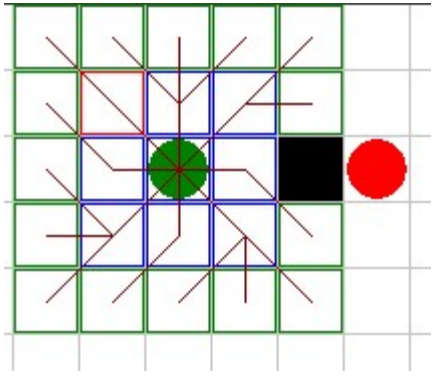
Breadth-first Search



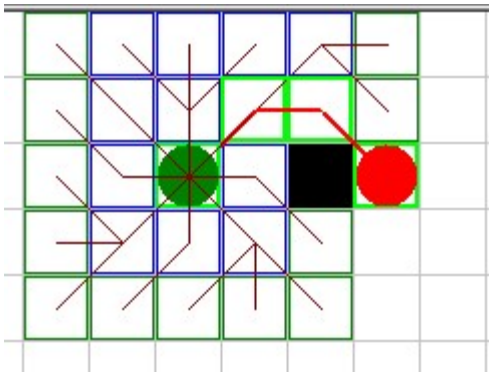
1



2



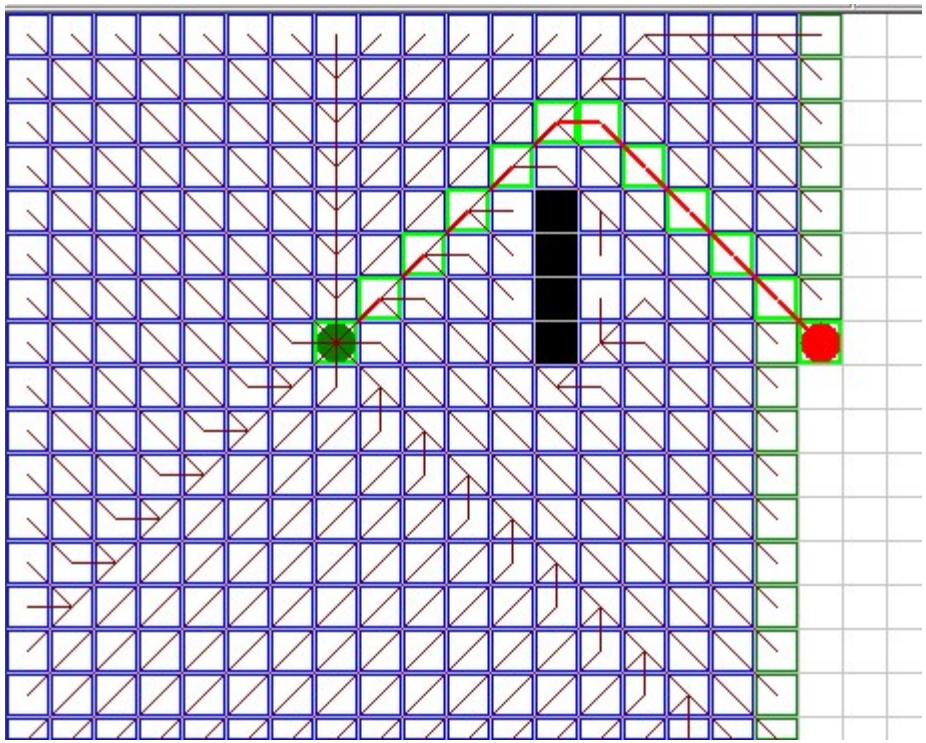
3



Goal!

Breadth-first Search

- Require much memory & computation time
- Not suitable for game usage



A* Search

- Popular & powerful search method
- Used in most strategy games
- Idea is to examine *first* those nodes which are more promising to get to goal state
- Need heuristic function to estimate how promising a node is

A* Search

- Overall score of a state is

  $f(node) = g(node) + h(node)$

  f(node) overall score of a node

  g(node) actual cheapest cost to that node

  h(node)

  heuristic estimate of reaching goal state from this node, e.g.

  horizontal distance + *vertical distance*

- Need to use a *priority queue Open* & *a list Closed*

```
s.g = 0 // s is the start node
s.h = GoalDistEstimate( s )
s.f = s.g + s.h
s.parent = null
push s on Open
while Open is not empty
    pop node n from Open // n has the lowest f
    if n is a goal node
            construct path
            return success
    for each successor n' of n
            newg = n.g + cost(n,n')
            if n' is in Open or Closed, and n'.g < = newg
                    skip
            n'.parent = n
            n'.g = newg
            n'.h = GoalDistEstimate( n' )
            n'.f = n'.g + n'.h
            if n' is in Closed
                    remove it from Closed
            if n' is not yet in Open
                    push n' on Open
    push n onto Closed
return failure // if no path found
```


A* Search

- Advantages
  1. Can give optimal results
  2. Can handle all geometry – concave & convex

- Problems
  1. Needs some twists so as to fit in fog-of-war techniques in most strategy games
  2. May have problem in scene with dynamic geometry
  3. If the map is divided into a large number of tiles, the memory required by A* is huge

Fog of War

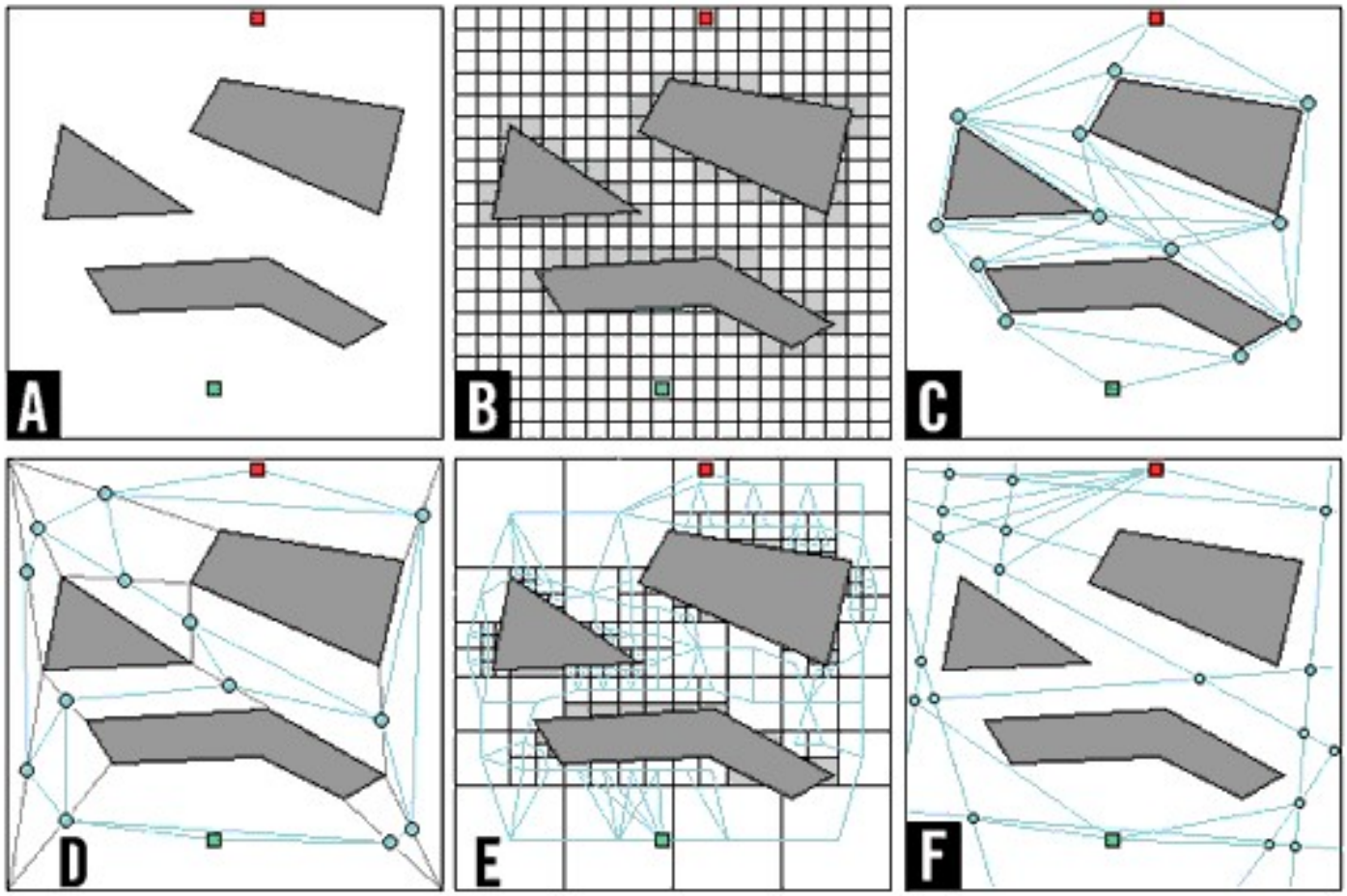
- covers everything that is not within the sight range of units or buildings
- Your units have no knowledge of geometry outside these regions

completely unexplored areas are fully black, while currently unobserved areas are covered in a grey shroud.

Region based A*

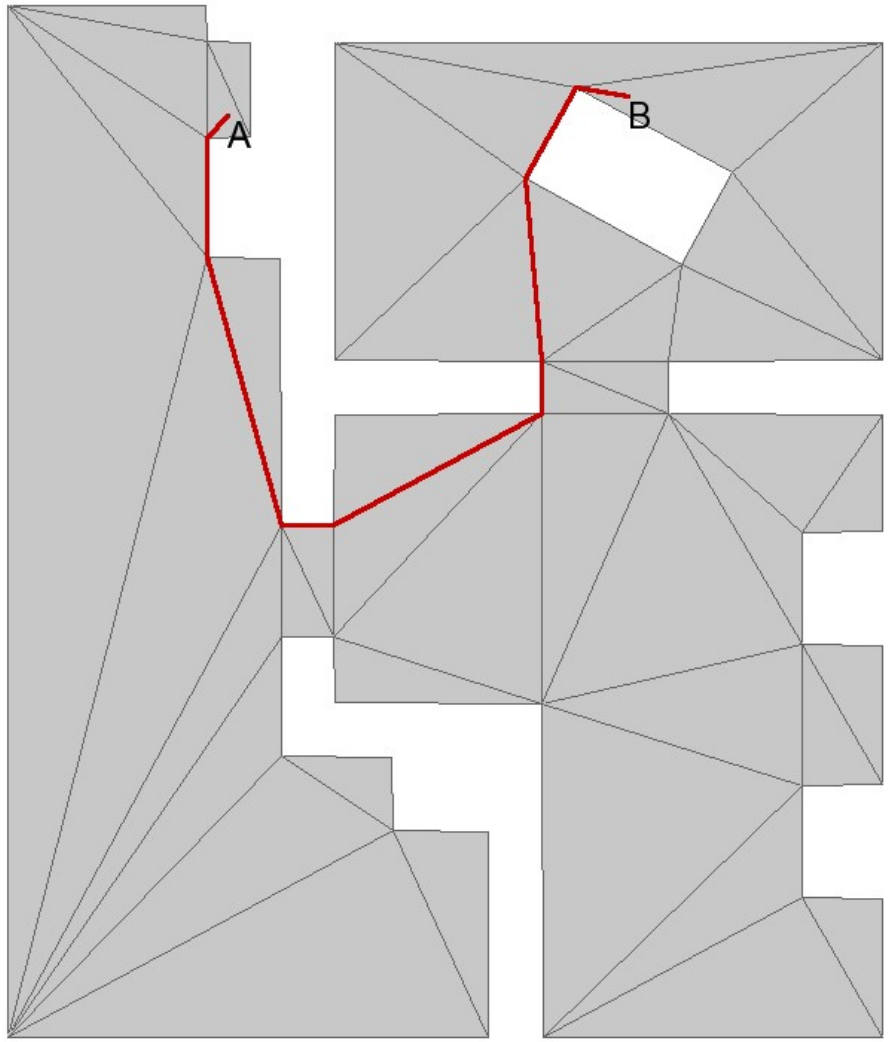
- Divide the map into a number of convex regions (zones, rooms)
- just use straight line path or crash and return within any region
- A* used in global map
- Used by most strategy games to create efficient path search

What if the world not discrete?



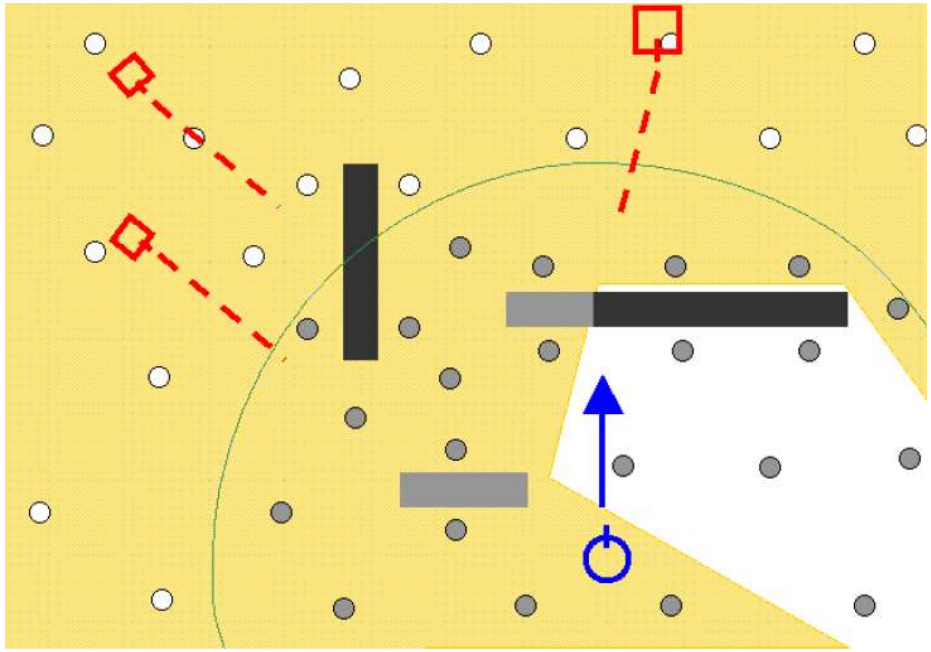
Navigation Mesh

- Application of A* in path finding
- For a path from A to B
  1. Locate the triangle A & B in
  2. Use A* to build a path from triangle A to B
- NavMesh constructed through analyzing level goemetry



What if the world 3D?

- 3D world brings complication to AI
- Dynamic world – changing geometry, need of AI to fight anywhere, against threats from any direction
- Even with scripts + waypoints will have problems in handling dynamic situations



Lines-of-fire & mobility in tactical shooters

Dynamic Tactics

- Take combat AI, which is most popular as example

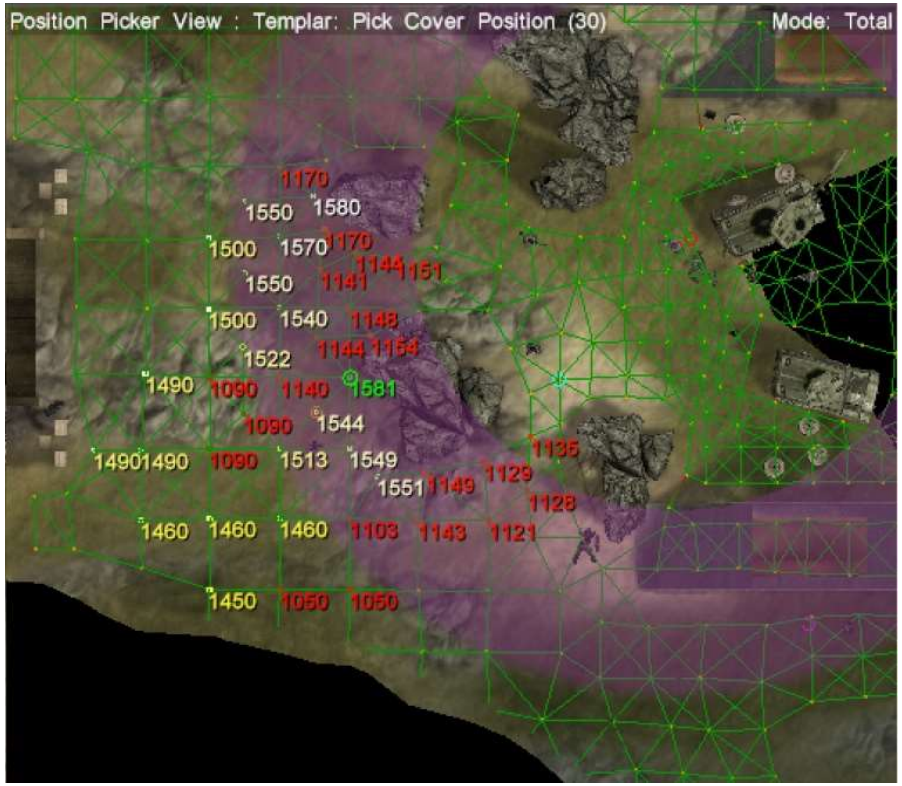
- Fire & maneuver
  1. Pick a destination
  2. Plan a path
  3. Move along path
  4. Arrive and start performing stationary action e.g. scanning, staying low etc.

Combat scene (Killzone)

Dynamic Tactics

- During modern 3D games, environment change a lot during path computation e.g. cover from multiple threats.

- A* is still good to use
- Static & dynamic information can be integrated into a single value as the heuristic function in A* => Position evaluation function



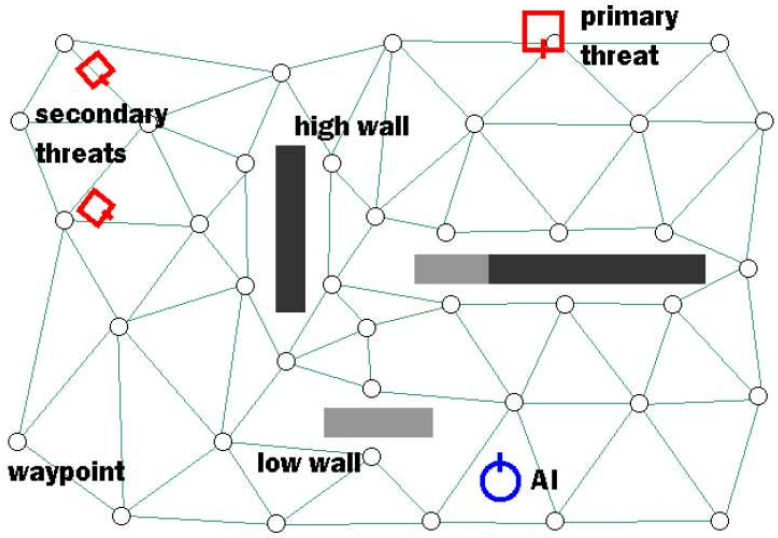
Position scores (Killzone)

Dynamic Tactics

- Tactical Position Picking
- AI will consider all positions within a radius around its current position

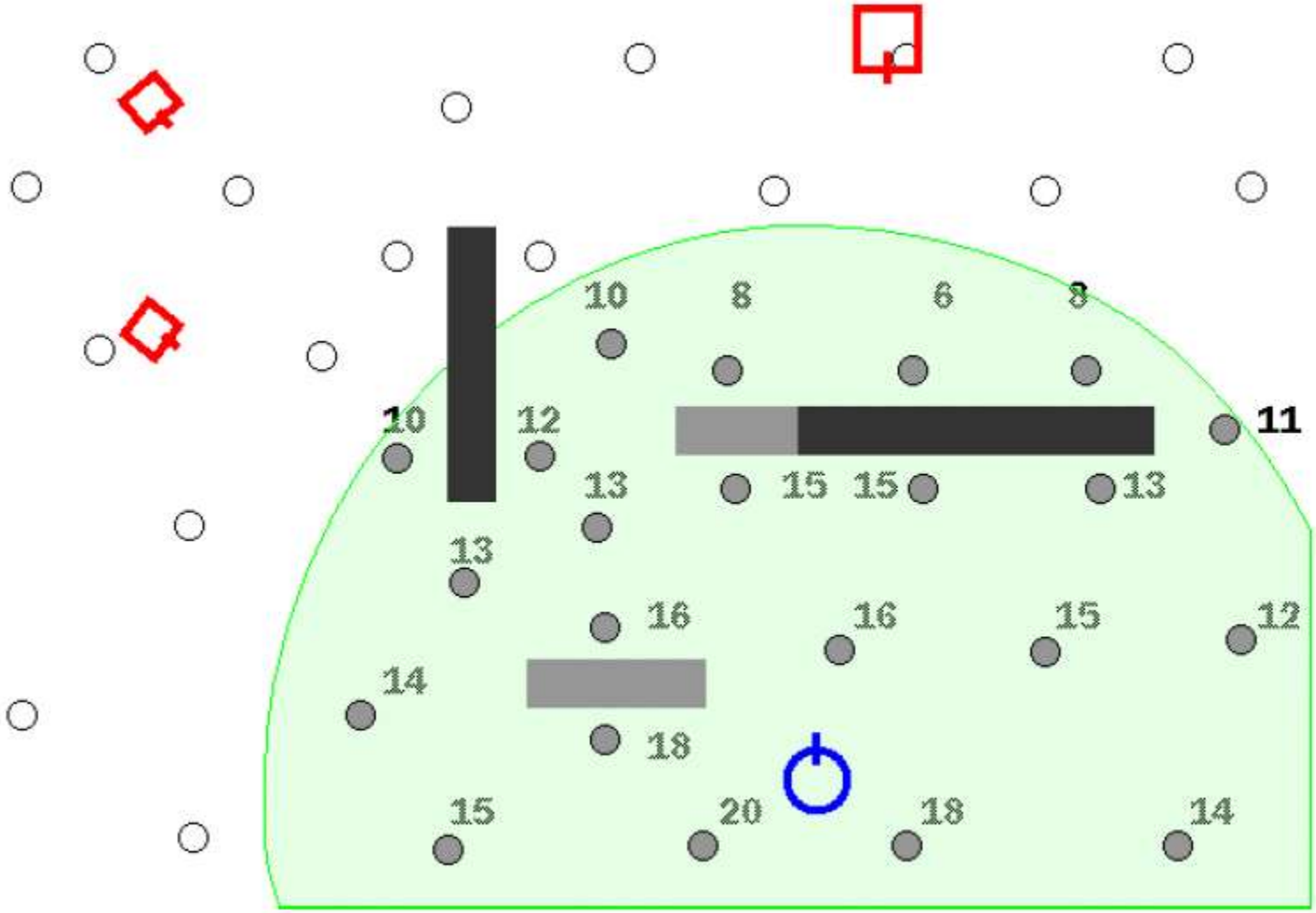
- Position evaluation functions will return all way points within the radius

- Consideration includes:
  - Proximity to current position
  - Cover from primary threat
  - Line of fire to primary threat, etc

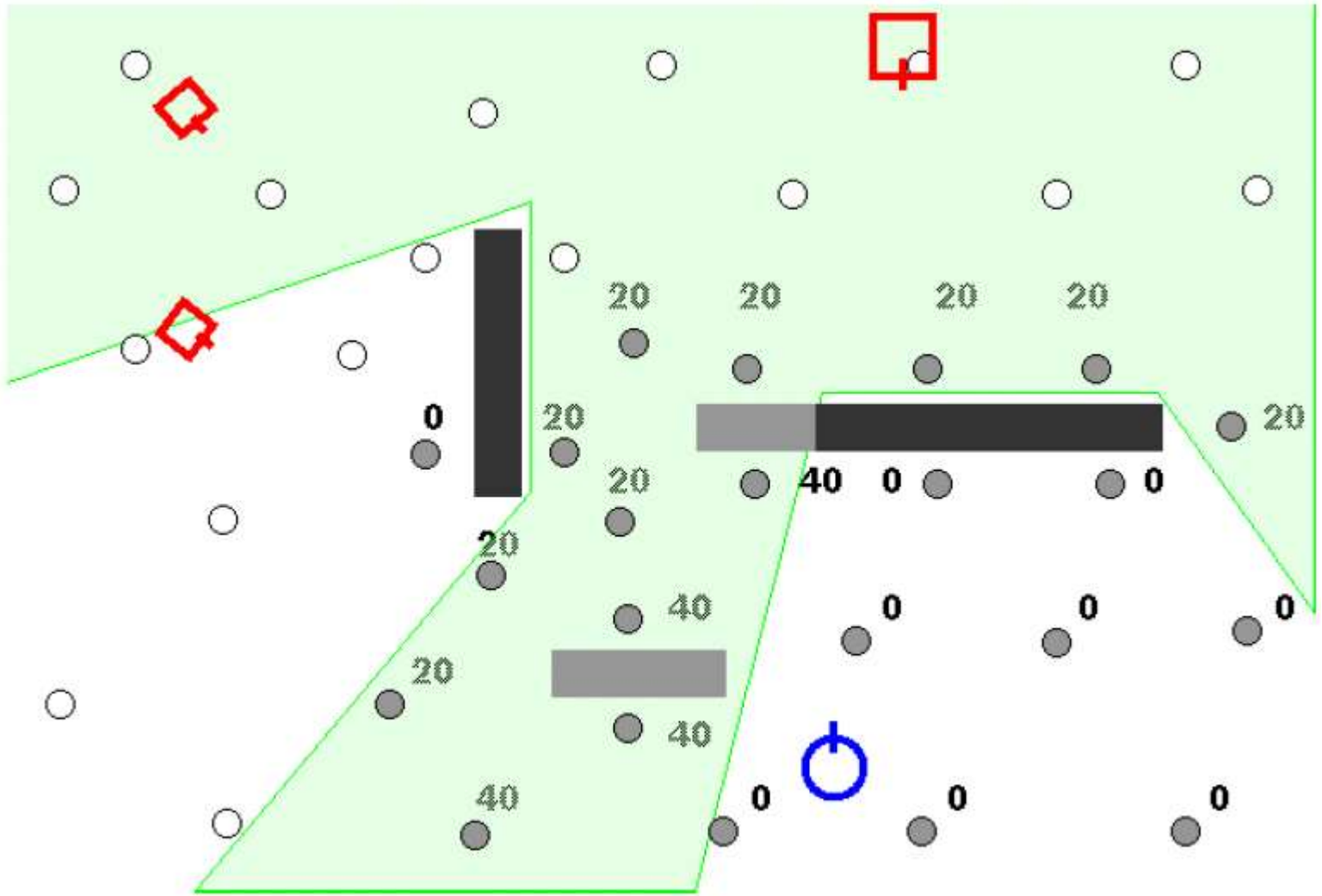


a. Initial situation with waypoint graph and legend.

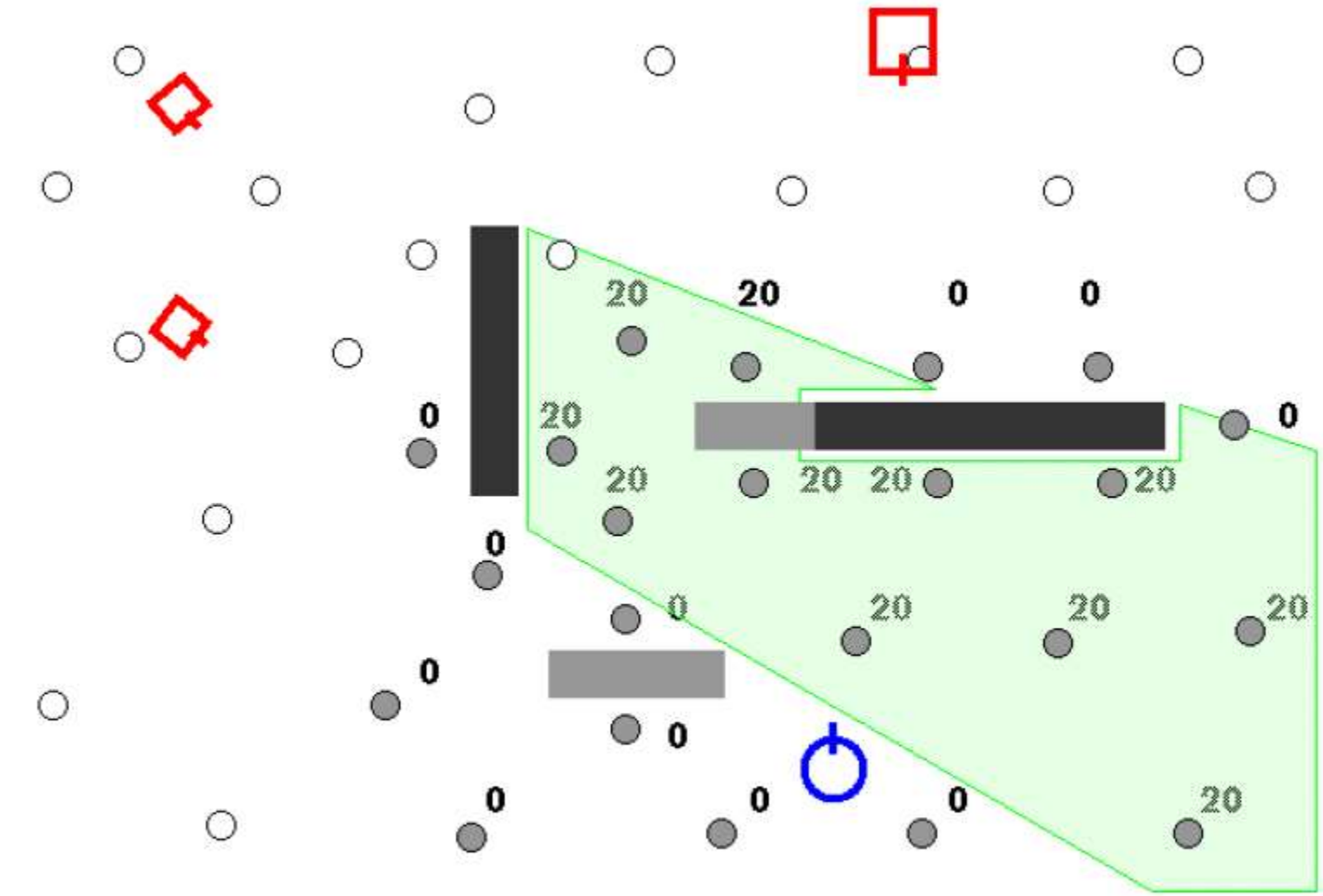
Killzone : tactical Position picking



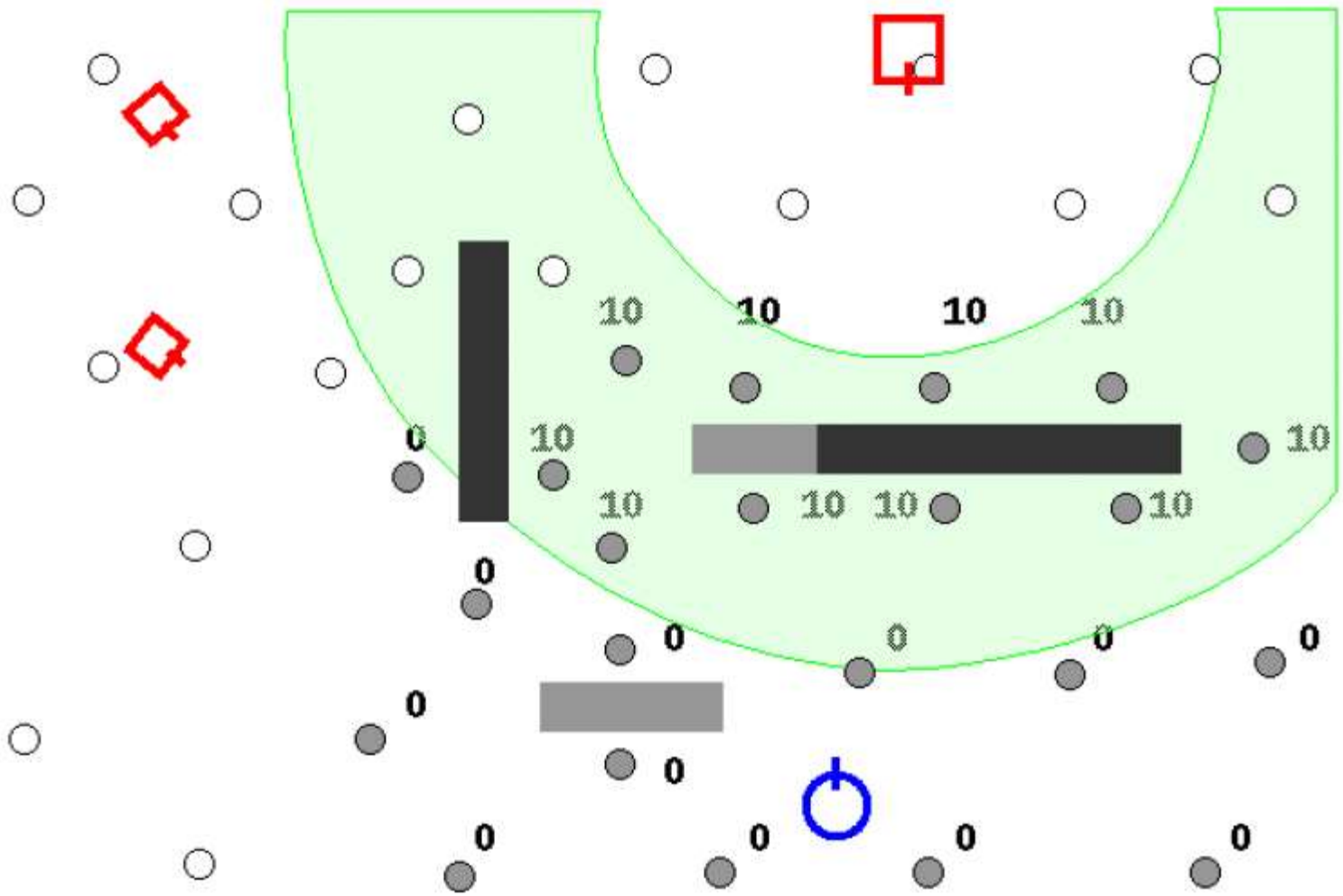
b. Selected nearby waypoints, annotated with proximity.



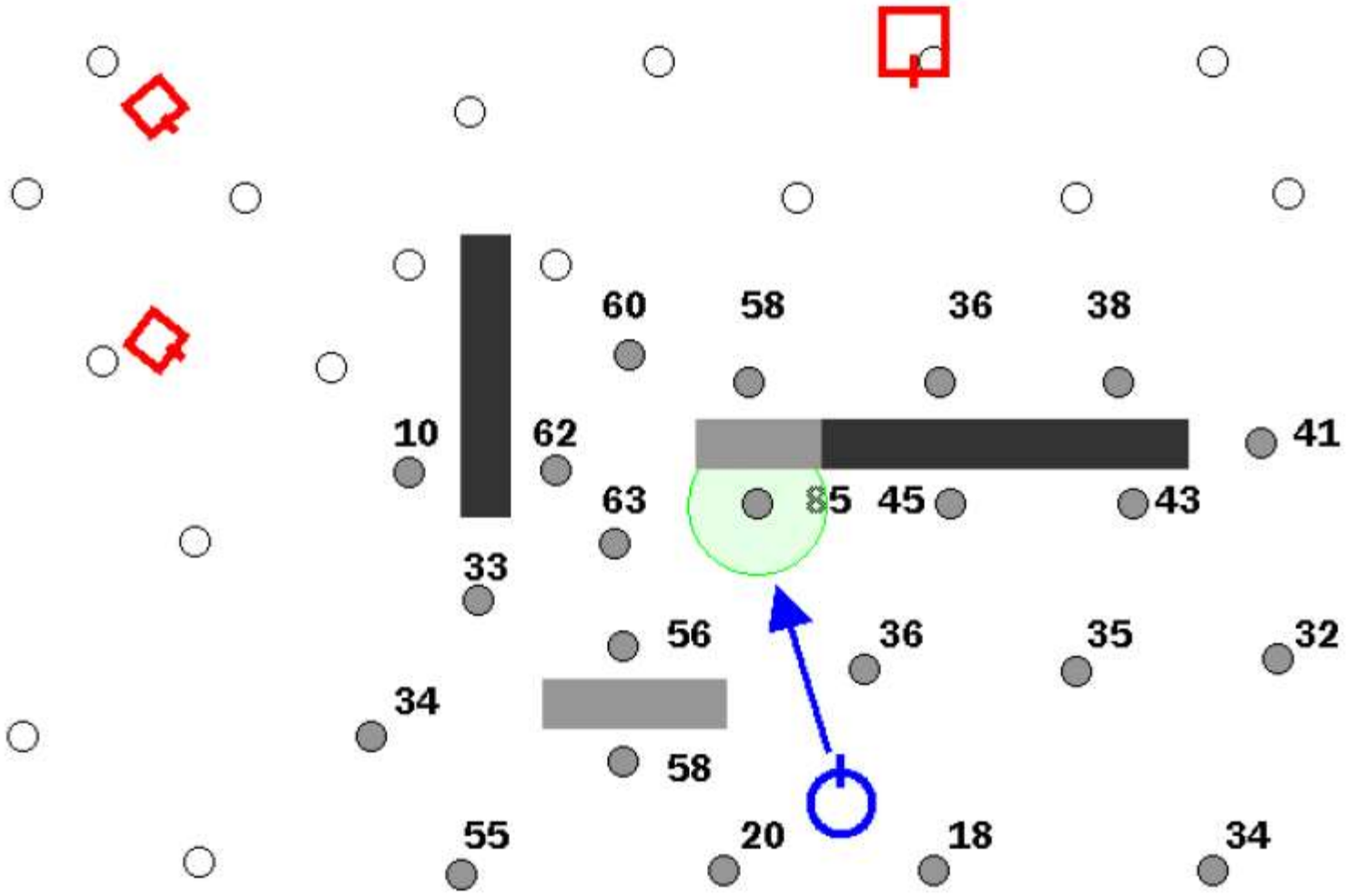
c. Annotations for positions with a line-of-fire to primary threat.



d. Annotations for positions with cover from the secondary threats.



e. Annotations for positions inside the preferred fighting range.



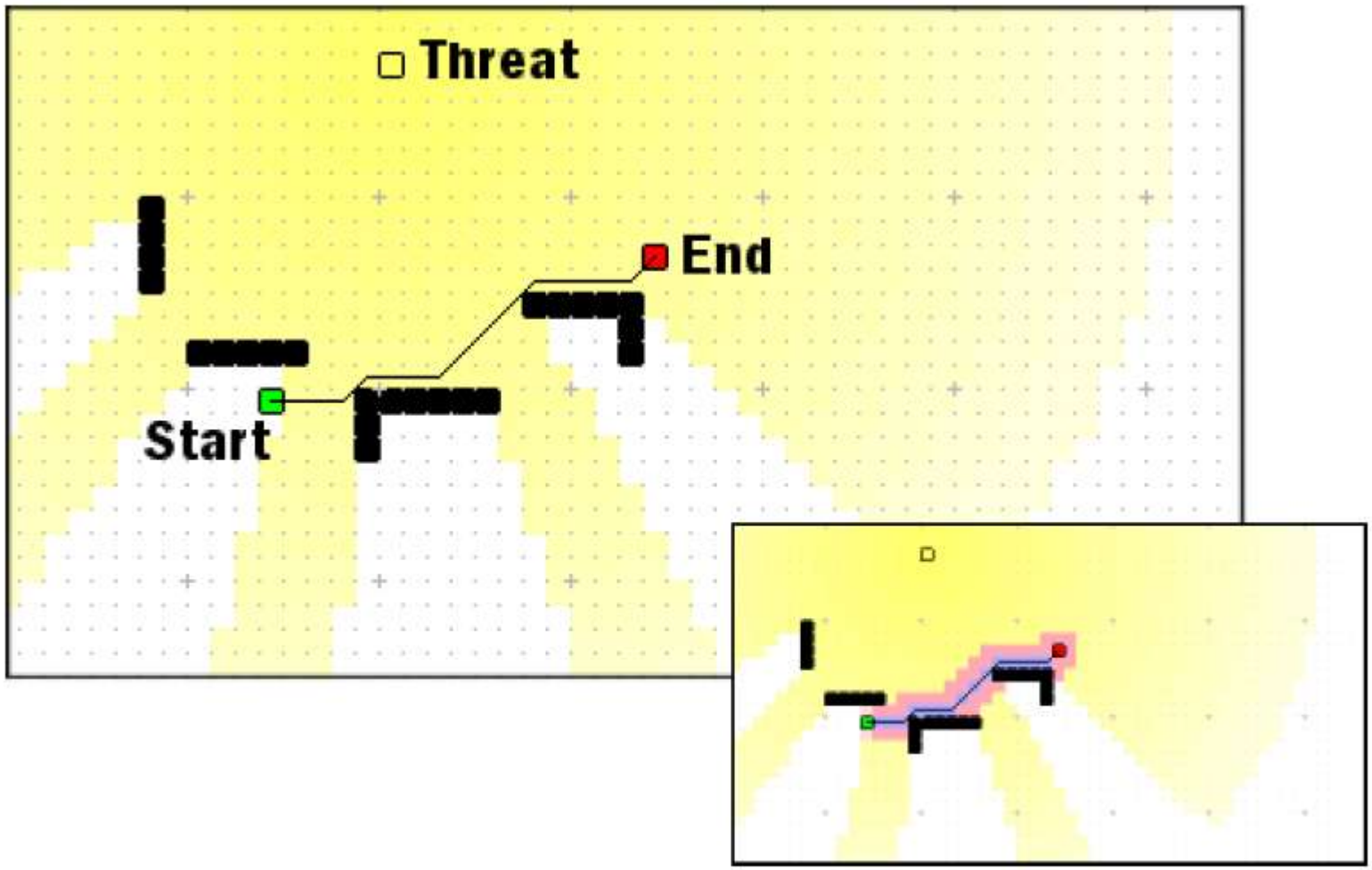
f. Adding up all the annotations yields the best attack position.

Tactical Path Finding

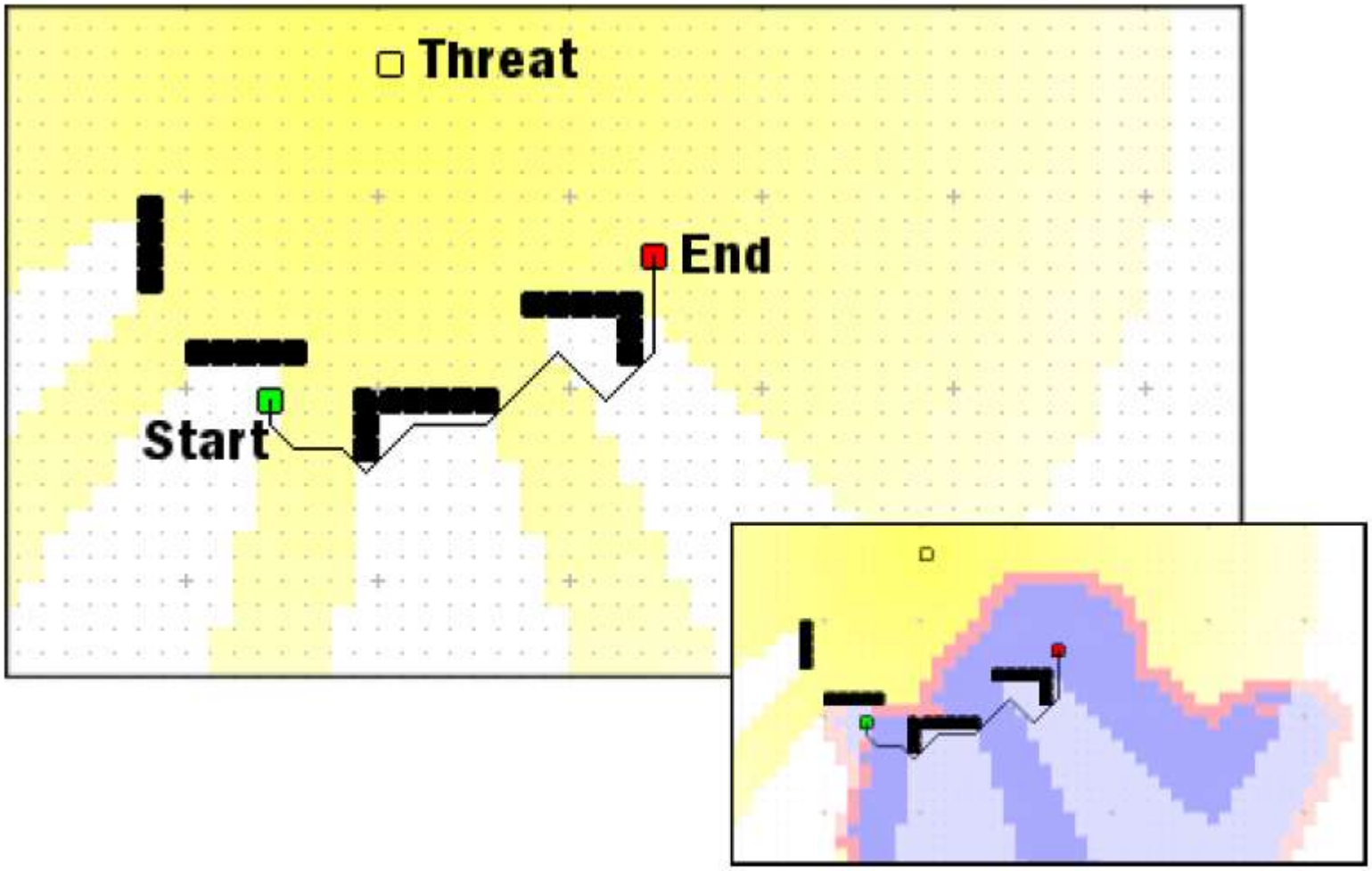
- can also incorporate position evaluation scores to make the planned path more responsive and tactical

- The cost function in A* can thus with added cost from position evaluation function

- Cost will be higher for crossing player's line of fire or to be visible to hostile guards



a. A shortest path ignoring the threat's line-of-fire. The inset shows how little terrain is explored by the shortest-path A* search.



b. A tactical path avoiding the threat's line-of-fire. The inset shows how much terrain the tactical A* search explores when it is not clamped to an area-of-operations

Usage of A*

A*	Navigation	Planning
Nodes:	NavMesh Polys	World States
Edges:	NavMesh Poly Edges	Actions
Goal:	NavMesh Poly	World State

Group Dynamics

- Movement patterns of groups of entities
- From simulating unorganized herd of animals to formation based march
- Difficult to use synchronization models as they cannot scale up to size of hundreds

Group Dynamics

- Each member evaluates its environment at every update cycle
- Can thus react to change in environment e.g. groups of swordsmen moving across a bridge or obstacles
- Useful in RTS and FPS

Boids algorithm

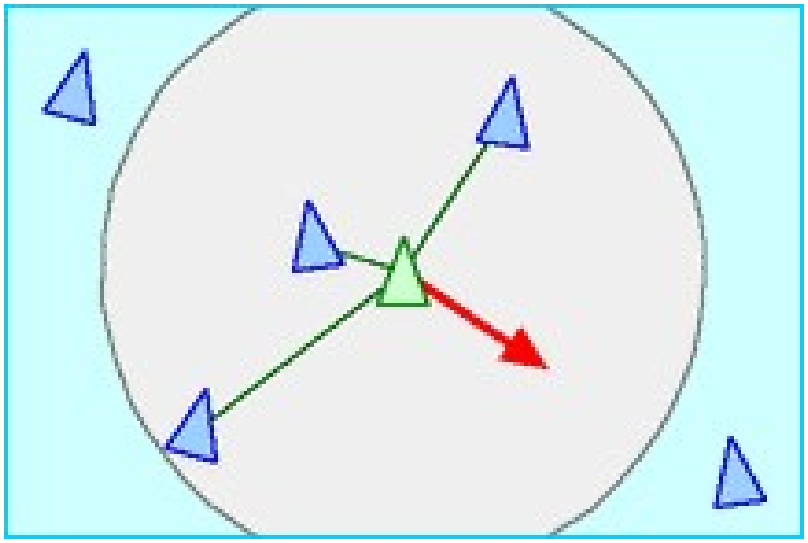
- Originally designed for special effect in movies
- Behavior of a group is governed by a small set of rules:

1. ***Separation***
2. ***Alignment***
3. ***Cohesion***

Simulated bird flocks movement

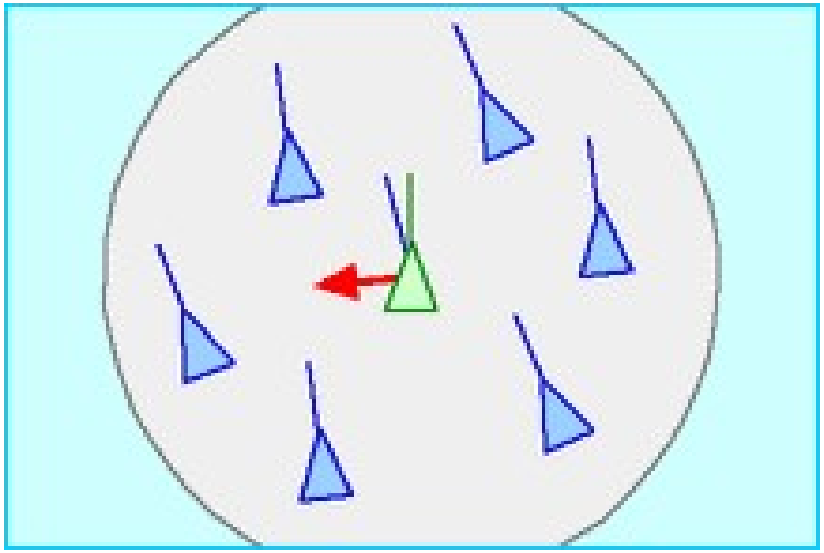
Separation

- steer to avoid crowding local flockmates



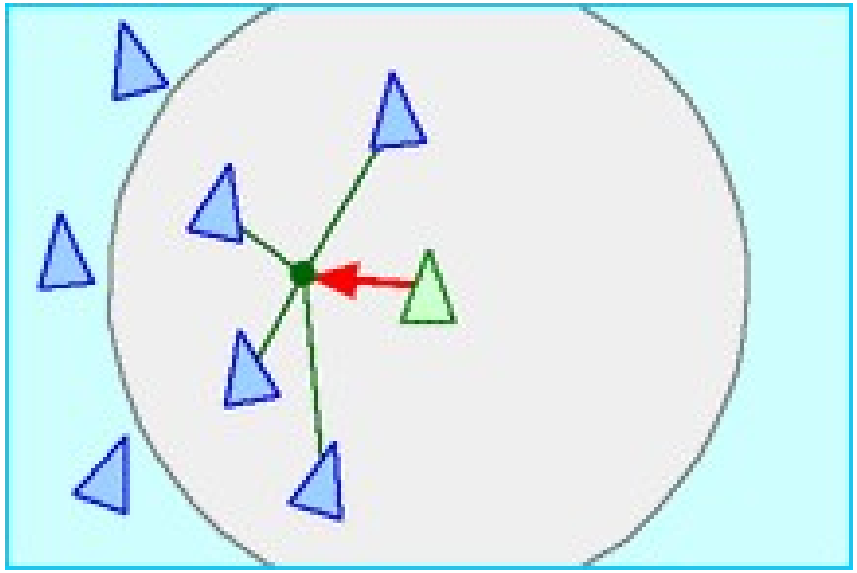
Alignment

- steer towards the average heading of local flockmates



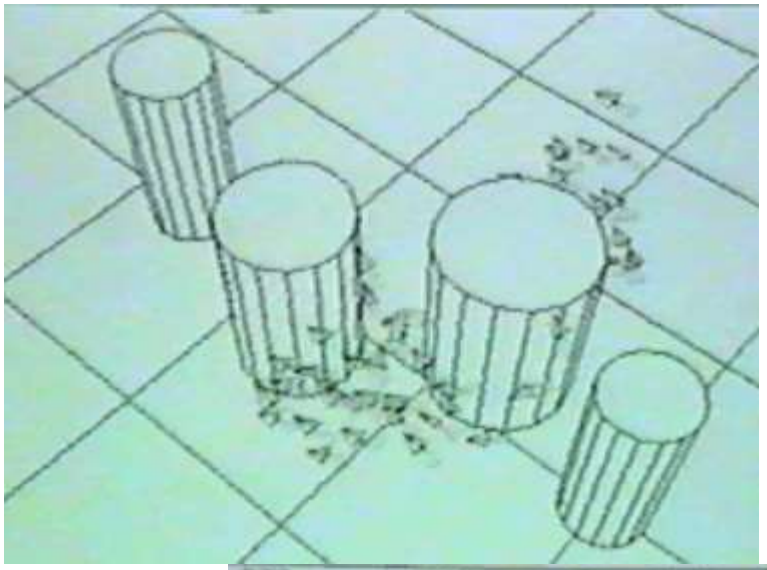
Cohesion

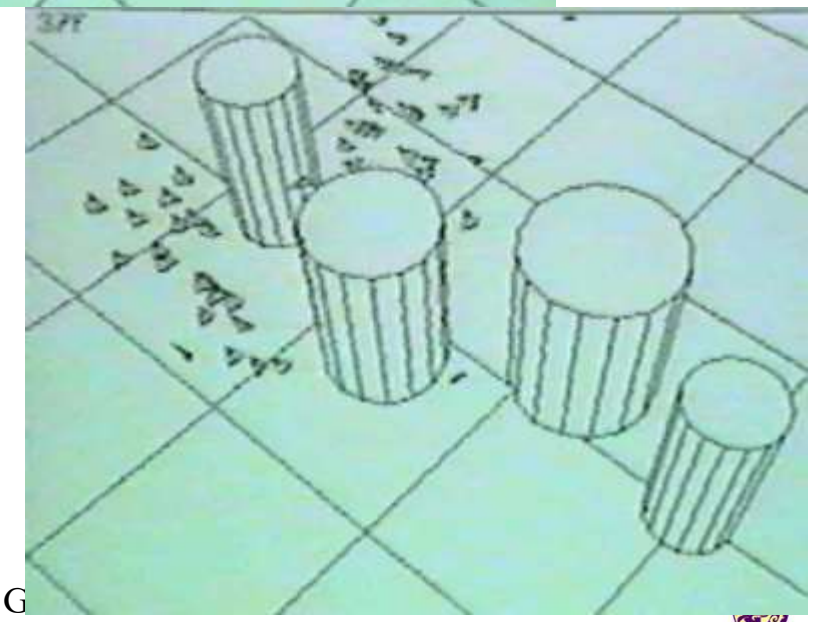
- steer to move toward the average position of local flockmates



Boids

- Can also be modeled as a combination of *attractive forces* towards the centre of flock and *repulsive forces* between individual member
- Prey and predator relationship can be modeled as a special case of *separation*
- Obstacle avoidance modeled also as extra type of entity that a strong repulsive forces apply to other member





Formation-based movement

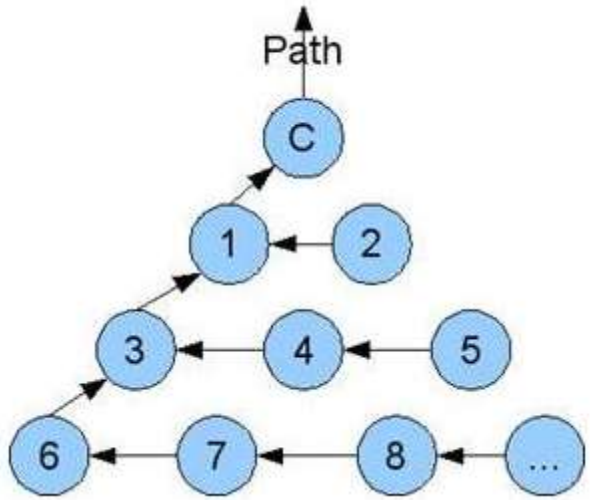
- cannot be modeled simply by boids as error in alignment would generate unwanted collisions/oscillations

- Save the loading of system when planning path

Age of Empire 4

Formation-based movement

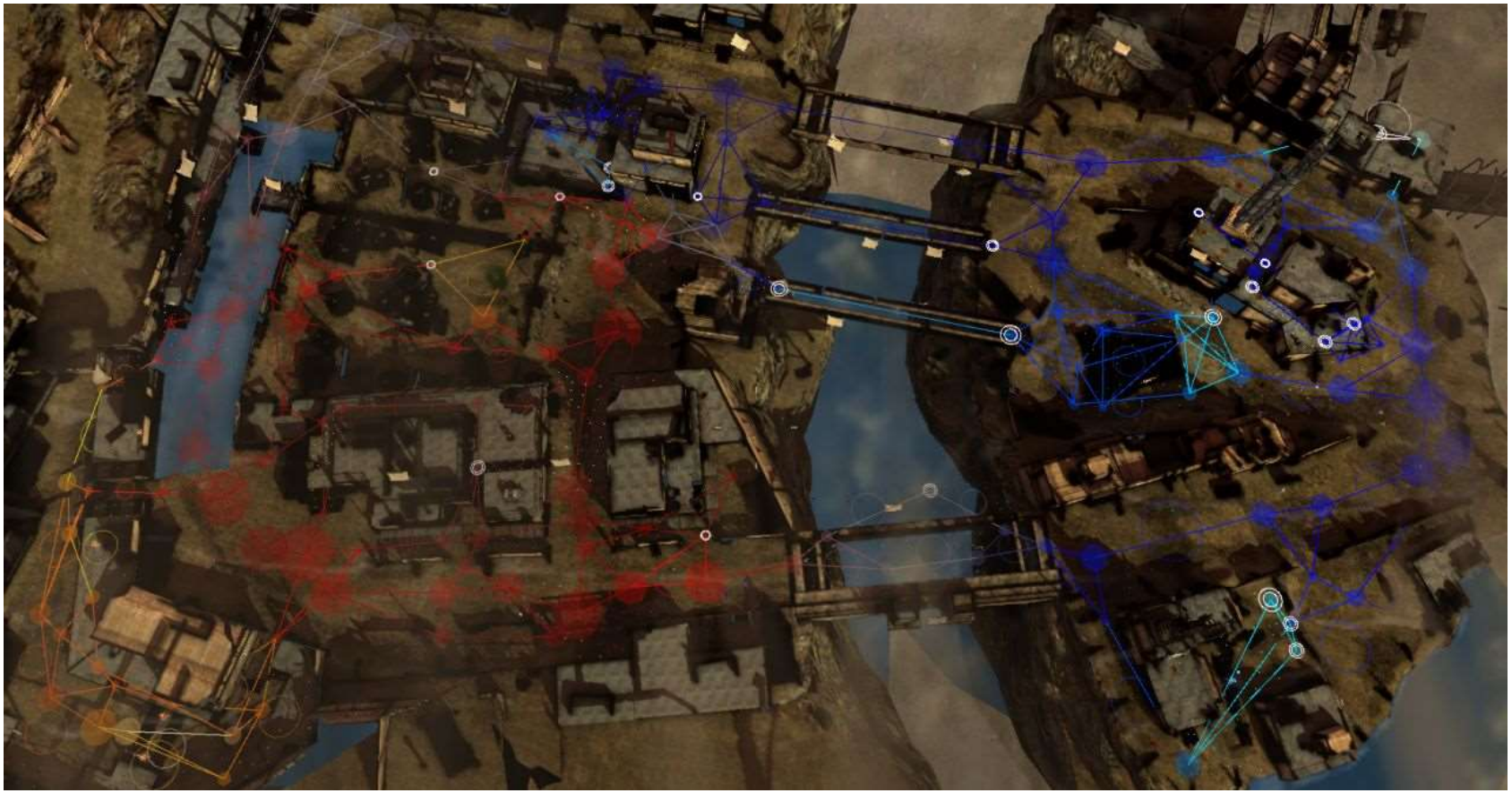
- Group captain set the path and all other units just follow
- Approach 1 – discard individual component and make whole squadron as a single entity
- Each unit just follow a set pace behind its leader, with group captain C moving as spearhead



Formation-based Movement

- Adapting boids algorithm by summation of *potential fields*
- One repulsive field : model separation
- One attractive field : keep an individual near the position supposed to occupy in the formation
- One repulsive field : perform collision voidance with obstacles

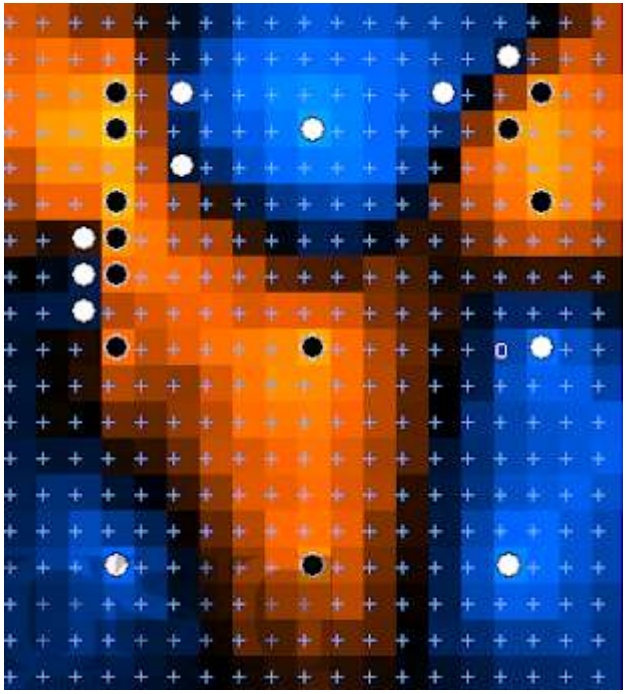
Military Analysis



- Influence map(IM) : allows performing queries for tactical analysis such as balance of power, frontlines, etc.

Influence Map

- 2D array of numeric values represent the influence of a certain variable e.g. soldiers' value
- Size depending on scenario
- Image processing techniques can then be applied to extract further information



http://gameschoolgems.blogspot.com/2009/12/influence-maps-i.html

Application of IM

- *Balance of power* : compute summation of whole IM. The side with positive value is winning

- *Frontline location*: zeros location

- *Breakability of enemy forces*: detect the two largest (or smallest) on enemy side. Trace a line between these two points and compare the value along the line for weakest point

Summary

- FSM, rule system as well as other techniques provide the foundation technologies for modern game
- To cope with increasing user expectation e.g. open world simulation, we still have to explore more novel techniques
- Simulation on crowds AI as well as applying machine learning as AI director helps making game play having more variety

References

- http://www.ml-class.org
- http://aigamedev.com/
- http://www.ai-blog.net/
- http://www.aiwisdom.com/
- http://en.wikipedia.org/wiki/Game_AI
- http://satirist.org/learn-game/
- http://christophermpark.blogspot.hk/2009/06/designing-emergent-ai-part-1.html
-