12-0: **Artifical Intelligence**

- AI in games is a huge field
  - Creating a believable world
    - Characters with their own appearnt goals and desires, especially in RPGs and open world games
    - Opponents that seem to think and plan
  - Simulating human players
    - Chess players, FPS “bots”, strategy game opponents, etc

12-1: **Most AI is Faked ...**

- ... which in unsurprising, since most *everything* is faked, if possible
- Don’t need to have intelligent enemies, just need to *appear* intelligent
- Surprisingly large quantity is done with Finite State Machines

12-2: **Finite state machines**

- Each entity has a number of states, that represent behaviors
  - Patrolling, advancing to a position, searching, running away, finding cover, etc
- Each behavior can be relatively simple
- Transitions between behaviors can be triggered by timers, scripting, “sensing” by entities, etc

12-3: **Case Study: Stealth shooter**

- Creating a stealth-based action game (Thief, Splinter Cell, Metal Gear Solid, etc)
  - Patrol state (traversing between waypoints)
  - Alerted state (simple search pattern)
  - Attacking state (advance towards player, attack)
- Each behavior is relatively simple, well-managed transitions between them (especially scripted transitions) can lead to very intelligent-seeming enemies. Add in some random audio cues, and the enemies can seem quite smart ...

12-4: **Pathfinding**

- One aspect of tradional AI that is commonly used in games is pathfinding
  - RTS units getting from home base to place they are attacking
  - Enemies attacking player in a maze-style game
  - Bots finding shortest route to powerups / other players / etc in FPSs
- First step: Simplifiying the problem

12-5: **Pathfinding**

- Navigating a real-life (or even complex simulated) enviornment is tricky
- Vastly simplify the search space, make it a standard CS-style graph

- Waypoint System
- Navigation Mesh

- 2D games (RTS, etc), can be easier – just use a grid

12-6: **Pathfinding**

- OK, so we've simplified the problem to searching for a path in a (potentially very complicated) graph
  - Verticies (places AI can go)
  - Edges (links between verticies, cost – often just a distance, can be mor complicated)
- How do we efficiently search the graph?

12-7: **Breadth-First Search**

- Examine all nodes that are 1 unit away
- Examine all nodes that are 2 units away
- ...
- Examine all nodes that are n units away

(Examples)

12-8: **Breadth-First Search**

- A few more wrinkes:
  - Searching a graph instead of a tree
  - Get to the same node in more than one way
  - Once we've found shortest path to a path to a node, don't need to consider any other paths

12-9: **Breadth-First Search**

- Maintain two data structures
  - "Open List" – search horizon
  - "Closed list" – nodes we've already found the shortest path to, don't need to examine again

12-10: **Breadth-First Search**

```
void BFS(Graph G, Vertex v) {

  Queue Q = new Queue();
  Closed = new ClosedList();

  Q.enquque(v);
  while (!Q.empty()) {
    nextV = Q.dequeue()
    if (v not in Closed)
     {
         Closed.Add(v);
         forach (Vertex neighbor adjacent to v in G)
             Q.enqueue(neighbor);
      }
    }
  }
}
```

12-11: **Breadth-First Search**

- Problem #1 with BFS:

- Assumes uniform edge cost
- Not actually true with most graphs we will be searching

- Solution?

12-12: **Best-first Search**

- Uniform-cost search
  - Store node *and cost to get to node* in queue
  - Use a priority queue instead of a standard queue
  - Always choose the cheapeast node to expand
    - “Expand” means examine children of node

12-13: **Uniform-Cost Search**

- Uniform-Cost Pseudocode

```
enqueue(initialState)
do
  node = prioroty-dequeue()
  if (node not in closed list)
     add node to closed list
     if goalTest(node)
        return node (potenially path as well)
     else
        children = successors(node)
        for child in children
           prioroty-enqueue(child, dist(child))
```

- $dist$ is the cost of the path from the initial state to the child node

(EXAMPLES!) 12-14: **Uniform-Cost Search**

- Problem with Uniform cost search
  - To find a goal that is 100 units away from the start, we examine *all* nodes that are 100 units away from the start
  - RTS example on board

- Make a minor change to Uniform cost serach, make it much more general

12-15: **Best-First Search**

```
enqueue(initialState)
do
  node = prioroty-dequeue()
  if (node not in closed list)
     add node to closed list
     if goalTest(node)
        return node (potenially path as well)
     else
        children = successors(node)
        for child in children
           prioroty-enqueue(child, f(child))
```

- $f(n)$ is a function that describes how “good” a node is

12-16: **Best-first Search**