



IMGD 1001 - The Game Development Process: Programming Practices; Artificial Intelligence

by

Robert W. Lindeman (gogo@wpi.edu)

Kent Quirk (kent_quirk@cognitoy.com)

(with lots of input from Mark Claypool!)



Outline

- Common Practices
- Artificial Intelligence

Common Practices: Version Control



- ❑ Database containing files and past history of them
- ❑ Central location for all code
- ❑ Allows team to work on related files without overwriting each other's work
- ❑ History preserved to track down errors
- ❑ Branching and merging for platform specific parts

Common Practices: Quality (1 of 3)



- ❑ *Code reviews* – walk through code by other programmer(s)
 - Formal or informal
 - "Two pairs of eyes are better than one."
 - Value is that the programmer is aware that others will read
- ❑ *Asserts*
 - Force program to crash to help debugging
 - ❑ Ex: Check condition is true at top of code, say pointer not NULL before continuing
 - Removed during release

Common Practices: Quality (2 of 3)



- Unit tests
 - Low level test of part of game (Ex: see if physics computations correct)
 - Tough to wait until very end and see if bug
 - Often automated, computer runs through combinations
 - Verify before assembling
- Acceptance tests
 - Verify high-level functionality working correctly (Ex: see if levels load correctly)
- Note, above are programming tests (i.e., code, technical)
 - Still turned over to testers that track bugs, do gameplay testing.

Common Practices: Quality (3 of 3)



- Bug database
 - Document and track bugs
 - Can be from programmers, publishers, customers
 - Classify by severity and priority
 - Keeps bugs from falling through cracks
 - Helps see how game is progressing

Common Practices: Pair (or "Peer") Programming



- Two programmers at one workstation
- One codes and tests, other thinks
 - Switch after fixed time
- Results
 - Higher-quality code
 - More bugs found as they happen
 - More enjoyable, higher morale
 - Team cohesion
 - Collective ownership

Outline



- Common Practices (done)
- Artificial Intelligence (next)

Introduction to AI

- ❑ Opponents that are challenging, or allies that are helpful
 - Unit that is credited with acting on own
- ❑ Human-level intelligence too hard
 - But under narrow circumstances can do pretty well (ex: chess and Deep Blue)
- ❑ Artificial Intelligence (around in CS for some time)

AI for CS different than AI for Games

- ❑ Must be smart, but purposely flawed
 - Loose in a fun, challenging way
- ❑ No unintended weaknesses
 - No "golden path" to defeat
 - Must not look dumb
- ❑ Must perform in real time (CPU)
- ❑ Configurable by designers
 - Not hard coded by programmer
- ❑ "Amount" and type of AI for game can vary
 - RTS needs global strategy, FPS needs modeling of individual units at "footstep" level
 - RTS most demanding: 3 full-time AI programmers
 - Puzzle, street fighting: 1 part-time AI programmer

AI for Games: Mini Outline



- Introduction (done)
- Agents (next)
- Finite State Machines

Game Agents (1 of 2)



- Most AI focuses around game agent
 - Think of agent as NPC, enemy, ally or neutral
- Loops through: sense-think-act cycle
 - Acting is event specific, so talk about sense+think
- Sensing*
 - Gather current world state: barriers, opponents, objects
 - Need limitations: avoid "cheating" by looking at game data
 - Typically, same constraints as player (vision, hearing range)
 - Often done simply by distance direction (not computed as per actual vision)
 - Model communication (data to other agents) and reaction times (can build in delay)

Game Agents (2 of 2)

Thinking

- Evaluate information and make decision
- As simple or elaborate as required
- Two ways:
 - Pre-coded expert knowledge, typically hand-crafted if-then rules + randomness to make unpredictable
 - Search algorithm for best (optimal) solution

Game Agents: Thinking (1 of 3)

Expert Knowledge

- Finite state machines, decision trees, ... (FSM most popular, details next)
- Appealing since simple, natural, embodies common sense
 - Ex: if you see enemy weaker than you, attack. If you see enemy stronger, then go get help
- Often quite adequate for many AI tasks
- Trouble is, often does not scale
 - Complex situations have many factors
 - Add more rules, becomes brittle

Game Agents: Thinking (2 of 3)



□ Search

- Look ahead and see what move to do next
- Ex: piece on game board, pathfinding (ch 5.4)

□ Machine learning

- Evaluate past actions, use for future
- Techniques show promise, but typically too slow
- Need to learn and remember

Game Agents: Thinking (3 of 3)



□ Making agents stupid

- Many cases, easy to make agents dominate
 - Ex: bot always gets head-shot
- Dumb down by giving "human" conditions, longer reaction times, make unnecessarily vulnerable

□ Agent cheating

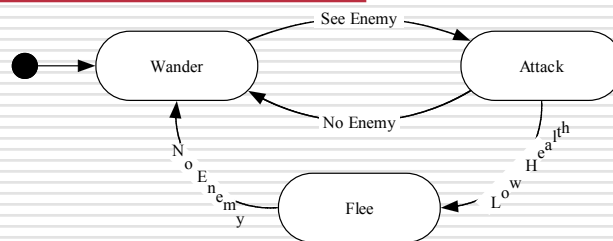
- Ideally, don't have unfair advantage (such as more attributes or more knowledge)
- But sometimes might, to make a challenge
 - Remember, that's the goal, AI lose in challenging way
- Best to let player know

AI for Games: Mini Outline



- Introduction (done)
- Agents (done)
- Finite State Machines (next)

Finite State Machines (1 of 2)



- Abstract model of computation
- Formally:
 - Set of states
 - A starting state
 - An input vocabulary
 - A transition function that maps inputs and the current state to a next state

Finite State Machines (2 of 2)

- Most common game AI software pattern
 - Natural correspondence between states and behaviors
 - Easy to diagram
 - Easy to program
 - Easy to debug
 - Completely general to any problem
- Problems
 - Explosion of states
 - Often created with ad-hoc structure

Finite-State Machines: Approaches

- Three approaches
 - Hardcoded (switch statement)
 - Scripted
 - Hybrid Approach

Finite-State Machine: Hardcoded FSM



```
void RunLogic( int * state ) {  
    switch( state ) {  
        case 0: //Wander  
            Wander();  
            if( SeeEnemy() ) { *state = 1; }  
            break;  
  
        case 1: //Attack  
            Attack();  
            if( LowOnHealth() ) { *state = 2; }  
            if( NoEnemy() ) { *state = 0; }  
            break;  
  
        case 2: //Flee  
            Flee();  
            if( NoEnemy() ) { *state = 0; }  
            break;  
    }  
}
```

Finite-State Machine: Problems with Switch FSM



1. Code is ad hoc
 - Language doesn't enforce structure
2. Transitions result from polling
 - Inefficient – event-driven sometimes better
3. Can't determine 1st time state is entered
4. Can't be edited or specified by game designers or players

Finite-State Machine: Scripted with alternative language

```
AgentFSM
{
    State( STATE_Wander )
    OnUpdate
        Execute( Wander )
        if( SeeEnemy ) SetState( STATE_Attack )
    OnEvent( AttackedByEnemy )
        SetState( Attack )
    State( STATE_Attack )
    OnEnter
        Execute( PrepareWeapon )
    OnUpdate
        Execute( Attack )
        if( LowOnHealth ) SetState( STATE_Flee )
        if( NoEnemy ) SetState( STATE_Wander )
    OnExit
        Execute( StoreWeapon )
    State( STATE_Flee )
    OnUpdate
        Execute( Flee )
        if( NoEnemy ) SetState( STATE_Wander )
}
```

Finite-State Machine: Scripting Advantages

1. Structure enforced
2. Events can be handed as well as polling
3. OnEnter and OnExit concept exists
4. Can be authored by game designers
 - Easier learning curve than straight C/C++

Finite-State Machine: Scripting Disadvantages



- ❑ Not trivial to implement
- ❑ Several months of development
 - Custom compiler
 - ❑ With good compile-time error feedback
 - Bytecode interpreter
 - ❑ With good debugging hooks and support
- ❑ Scripting languages often disliked by users
 - Can never approach polish and robustness of commercial compilers/debuggers
 - Though, some are getting close!

Finite-State Machine: Hybrid Approach



- ❑ Use a class and C-style macros to approximate a scripting language
- ❑ Allows FSM to be written completely in C++ leveraging existing compiler/debugger
- ❑ Capture important features/extensions
 - OnEnter, OnExit
 - Timers
 - Handle events
 - Consistent regulated structure
 - Ability to log history
 - Modular, flexible, stack-based
 - Multiple FSMs, Concurrent FSMs
- ❑ Can't be edited by designers or players
- ❑ Kent says: "Hybrid approaches are evil!"