Piazza Registration

- Automatic registration for enrolled students.

- Those not enrolled:
  - Please sign up individually.
  - My office hours or see Bryan.

- https://piazza.com/ucsc/fall2013/cmps146
Readings

- First three will be posted right after class.
  - First lecture: 19-35
  - Today's lecture: 293-306

- Who has books?
Pac-Man AI Competition Information

http://www.pacman-vs-ghosts.net/
DRC Accommodations

If you qualify for classroom accommodations because of a disability, please get an Accommodation Authorization from the Disability Resource Center (DRC) and submit it to the instructor in person outside of class (e.g., office hours) within the first two weeks of the quarter. Contact DRC at 459-2089 (voice), 459-4806 (TTY), or http://drc.ucsc.edu for more information on the requirements and/or process.
Decision Making AI

Internal Knowledge

External Knowledge

Internal changes

Decision Maker

External changes

Action Request
The AI Model

- Execution Management
- Group AI
  - Strategy
- Character AI
  - Decision Making
  - Movement
- World Interface
  - Animation
  - Physics
- Content Creation
  - Scripting
Lots of branching code:
  Difficult to author
  Difficult to maintain
  Difficult to debug
  Hard to insert or modify behaviors
  Hard to reason temporally
DIY: Simple Example

If visible and distance is less than 10, then attack.

If visible and flanking, then move.

If visible and not flanking, then attack.

If not visible and is audible, then creep.
Simple Example

If (visible) {
    if (distance < 10) {
        attack();
    } else {
        if (flank){
            move();
        } else {
            Attack();
        }
    }
} else if (audible){
    creep();
}
SimCity/Micropolis Source Code Example

SimCity/Micropolis Source Release

http://www.donhopkins.com/home/micropolis/

GNU GPL
Spaghetti
Complexity
Decision Trees

- Easy to understand
- Easy to implement
- Fast execution
- First example of knowledge representation (abstracting decision making from raw code)
Example Decision Tree

Is enemy visible?
  No
  Is enemy audible?
    Yes
    Creep
    No
    Is enemy < 10m away?
      Yes
      Attack
      No
      Attack
      Is enemy on flank?
        Yes
        Move
        No
        Creep
If (visible) {
  if (distance < 10) {
    attack();
  } else {
    if (flank) {
      move();
    } else {
      Attack();
    }
  }
} else if (audible) {
  creep();
}
If (visible) {
    if (distance < 10) {
        attack();
    } else {
        if (flank){
            move();
        } else {
            Attack();
        }
    }
} else if (audible) {
    creep();
}
If (visible) {
    if (distance < 10) {
        attack();
    } else {
        if (flank){
            move();
        } else {
            Attack();
        }
    }
} else if (audible){
    creep();
}
If (visible) {
    if (distance < 10) {
        attack();
    } else {
        if (flank){
            move();
        } else {
            Attack();
        }
    }
} else if (audible){
    creep();
}
Benefits of Decision Tree Representation

- Provides a simple abstraction above raw code
- Easier to understand and modify
- Supports team development
- Fast
Representing And and Or

IF A and B then 1 else 2

IF A or B then 1 else 2
Pseudo Code for Decision Tree

```python
class DecisionTreeNode:
    def makeDecision()

class Action (DecisionTreeNode):
    def makeDecision():
        return this

class FloatDecision (Decision):
    minValue
    maxValue
    def getBranch():
        if maxValue >= testValue >= minValue: return trueNode
        else: return falseNode
    def makeDecision():
        branch = getBranch()
        return branch.makeDecision()```
DIY: Red Ghost Decision Tree
Creating a Decision Tree

visible = new Boolean…
audible = new Boolean…

close = new MinMax…
flank = new Boolean…

attack = new Attack…
move = new Move…
creep = new Creep…

visible.yesNode = close
visible.noNode = audible

Audible.yesNode = creep
close.yesNode = attack
close.noNode = flank

Support with tool chain (e.g. graphical editor, XML…)
Modifying a Decision Tree

visible = new Boolean...
audible = new Boolean...
close = new MinMax...
flank = new Boolean...

attack = new Attack...
move = new Move...
creep = new Creep...

visible.yesNode = close
visible.noNode = audible
Audible.yesNode = creep
close.yesNode = attack
close.noNode = flank
Multiple Branches

class MultiDecision (DecisionTreeNode) :
   daughterNodes
   testValue

def getBranch():
   return daughterNodes[testValue]
Random Decisions

```python
class RandomDecision(Decision):
    lastFrame = -1
    lastDecision = false

def test() :
    if frame() > lastFrame + 1:
        lastDecision = randomBoolean()
        lastFrame = frame()
    return lastDecision

def getBranch() :
    decision = test()
    if decision :
        return trueNode
    else:
        return falseNode
```

Under attack?
- Yes
  - Defend
- No
  - Flip a coin
    - H
      - Patrol
    - T
      - Stand still
Authoring Aside: Perception of Random Game

Player → Game → Designer/programmer

Goals, Beliefs, Emotions

Diagram showing the interaction between the player, game, and designer/programmer.
Test nodes:
   Fast (constant time) - basic operations on primitives.
   Long processing times operations (like raycasting) to be avoided.

Action nodes:
   Standard game code performance optimizations (memory locality,
   avoid virtual function calls, etc.).

Tree shaping:
   Balance tree - makes binary trees $O(\log n)$.
   Heavy tests at the end of the tree – minimize hits on these nodes.
   Paths commonly traversed should be short.
Performance

- Tests are generally quite fast
  - Need to be careful that tests aren’t making expensive calls to the game engine
  - To decouple tree from engine changes, abstract values that can be sensed

- In a balanced tree, decision requires $O(\log_2 N)$ tests
  - Worst case performance based on depth
  - But may want to make common behaviors require fewer decisions
  - May want expensive decisions deeper