Nuts and Bolts: Modular AI from the Ground Up

Kevin Dill
Motivation

Chris Hecker
GDC 2008
http://chrishecker.com/Structure_vs_Style

structure
vs
style

Chris Hecker  Maxis / EA  chrishecker.com
Motivation

The Heartbreaking Beauty of the Texture Mapped Triangle

Chris Hecker
GDC 2008
http://chrishecker.com/Structure_vs_Style
The Game AI Architecture (GAIA)

• Lockheed Martin's modular architecture
• Used across 6 very different projects
  o character AI / sniper AI / strategic simulation / flight simulator
• Integrated into multiple engines
  o Gamebryo / Real World
  o VBS2
  o Havok
  o Unity
  o JSAF (Joint Semi-Automated Forces) simulator
  o (in progress) TES (Tactical Environment Simulation) simulator
  o (in progress) Web Server-based integration
Agenda

• What is Modular AI?
• Common Conceptual Abstractions
• Sniper Example
• Implementation
• Parting Thoughts
The Big Idea

- Level of granularity
  - "Bite-sized pieces"
  - Single human concept
- For example:
  - How far away is he?
  - How long have I been doing this?
  - Do I have any grenades?
  - I want to move over there
  - I want to shoot at that guy
Bite-Sized Pieces

• Conceptual Abstractions
  o Consideration
  o Action

• Modular Components
  o Distance Consideration
  o Move Action

• Implementation vs. Configuration
Agenda

- What is Modular AI?
- Common Conceptual Abstractions
- Sniper Example
- Implementation
- Parting Thoughts
Reasoners

- The thing that makes decisions
  - Utility-Based
  - Rule-Based
  - Sequence
  - ...

```cpp
class AIReasonerBase : public AIBase
{
public:
  virtual bool Init(const AICreationData& creationData);
  // Run any reasoner-specific sensors.
  void Sense(AIContext* pContext);

  // Think() is the meat of the reasoner. It is typically called every
  // frame. It handles selecting an option for execution, deselecting
  // the previous option when the selected option changes, and then
  // updating the selected option so that its actions can execute
  virtual void Think(AIContext* pContext);
};
```
Considerations

- Evaluate a single aspect of the current situation
  - Distance
  - Execution History
  - Picker
  - ...

```cpp
class AIConsiderationBase
{
public:
  virtual void Init(AICreationData& creationData) = 0;

  // Evaluate the situation and determine how “good” this option is.
  // Store the results in m_Weights. Access them with GetResults().
  virtual void Calculate() = 0;
  const AIWeightValues& GetResults() { return m_Weights; }

  // Some functions need to know when the associated option is
  // selected/deselected (for example, to store timing information).
  virtual void Select(AIContext* /*pContext*/) {}
  virtual void Deselect(AIContext* /*pContext*/) {}

protected:
  AIWeightValues m_Weights;
};
```
Actions

- What to do when a particular option is selected
  - Move
  - Fire Weapon
  - Subreasoner
  - ...

```cpp
class AIActionBase
{
public:
    virtual void Init(AICreationData& creationData) = 0;

    // Called when the action starts/stops execution.
    virtual void Select() {}
    virtual void Deselect() {}

    // Called every frame while we're selected.
    virtual void Update() {}

    // Check whether this action is finished executing. Some actions (such
    // as a looping animation) are always considered to be done, but others
    // (such as moving to a position) can be completed.
    virtual bool IsDone() { return true; }
};
```
Targets

- Represents a position and (optionally) an entity
  - Fixed Position
  - Named Entity
  - Controlled Entity
  - ...

```cpp
class AITargetBase
{
public:
  virtual bool Init(const AICreationData& cd);

  // Get the target's position
  virtual const AIVectorBase& GetPosition() const = 0;

  // Get the entity associated with this target (if any)
  virtual AIEntity* GetEntity() const { return NULL; }
  virtual bool HasEntity() const { return false; }
};
```
Weight Functions

- Convert from an input (e.g. Float, Boolean, etc.) to weight values
  - Boolean
  - Float Sequence
  - Simple Curve
  - ...

```cpp
class AIWeightFunctionBase
{
public:
  virtual bool Init(const AICreationData& cd) = 0;

  // Weight functions can deliver a result based on the input of
  // a bool, int, or float. By default bool and float both throw
  // an assert, and int calls float.
  virtual const AIWeightValues& CalculateBool(bool b);
  virtual const AIWeightValues& CalculateInt(int i);
  virtual const AIWeightValues& CalculateFloat(float f);

  // Some functions need to know when the associated option is
  // selected/deselected (for example, to readjust random values).
  virtual void Select() {};
  virtual void Deselect() {};
};
```
Regions

- Represents a region of space with an inside and an outside
  - Circle
  - Rectangle
  - Polygon
  - ...

```cpp
class AIRegionBase
{
public:
    virtual bool Init(const AICreationData& cd);

    // Test if the passed in location is within the geometry.
    virtual bool InRegion(const AIVectorBase& position) const = 0;

    // Get a random position within the geometry
    // NOTE: IT IS POSSIBLE FOR THIS TO FAIL!! It returns success.
    virtual bool GetRandomPosition2d(AIVectorBase& outVal) const = 0;
};
```
Agenda

• What is Modular AI?
• Common Conceptual Abstractions
• Sniper Example
• Implementation
• Parting Thoughts
Sniper

- Periodically (every minute or two) takes a shot at the enemy
  - Not if there is no line of retreat
  - Decrease priority with each additional shot
Sniper - The “Take A Shot” Option

Take A Shot

Considerations
• Execution History (Timer)
• Picker (Select Target)
• Picker (Line of Retreat)
• Integer Variable (Number of Shots)

Actions
• Write Blackboard (# Shots Fired)
• Fire at Target

<Option Type="ConsiderationAndAction" Comment="Take A Shot">
  <Considerations>
    <Consideration Type="ExecutionHistory">
      <StoppedWeightFunction Type="FloatSequence">
        <Entries>
          <Entry Min="60" Max="120" Veto="true"/>
        </Entries>
      </StoppedWeightFunction>
    </Consideration>
    <Consideration Type="Global" Name="PickTarget"/>
    <Consideration Type="Global" Name="CheckRetreat"/>
    <Consideration Type="IntegerVariable" Variable="NumShots">
      <WeightFunction Type="BasicCurve"/>
    </Consideration>
  </Considerations>
  <Actions>
    <Action Type="UpdateIntegerVariable" Variable="NumShots" UpdateType="Increment"/>
    <Action Type="Global" Name="FireAtTarget"/>
  </Actions>
</Option>
What Does This Buy Me?

- Appropriate level of abstraction
  - Enter ~6 values vs. a couple hundred lines of code
  - Those values are the relevant ones
- Broad reuse of both components (code) and behavior (XML)
  - Implement once
  - Fewer bugs
  - More mature code (better tested, more feature-rich)
- The Bottom Line: Developer Flow

```xml
<Consideration Type="ExecutionHistory">
  <StoppedWeightFunction Type="FloatSequence">
    <Entries>
      <Entry Min="60" Max="120" Veto="true"/>
    </Entries>
    <Default Veto="false"/>
  </StoppedWeightFunction>
</Consideration>
```
Agenda

• What is Modular AI?
• Common Conceptual Abstractions
• Sniper Example
• Implementation
• Parting Thoughts
Polymorphism

- Defines the interface
- Decouples interface from implementation

class AIConsiderationBase
{
public:
    virtual void Init(AICreationData& creationData) = 0;

    // Evaluate the situation and determine how “good” this option is.
    // Store the results in m_Weights. Access them with GetResults().
    virtual void Calculate() = 0;
    const AIWeightValues& GetResults() { return m_Weights; }

    // Some functions need to know when the associated option is
    // selected/deselected (for example, to store timing information).
    virtual void Select(AIContext* /*pContext*/) {}
    virtual void Deselect(AIContext* /*pContext*/) {}

protected:
    AIWeightValues m_Weights;
};
Factories

• **Input:** AICreationData
  - An XML node
  - Context data (the blackboard, the parent entity, the parent option, etc.)
• **Output:** an object of the appropriate subtype
• **E.G.** AIConsiderationFactory

```cpp
template<class T>
class AIFactoryBase
{
public:
    T* Create(AICreationData& creationData);

    // Add a custom constructor. Takes ownership of the constructor.
    void AddConstructor(AIConstructorBase<T>* pConstructor);
};
```
Factories - Bells & Whistles

• Constructors
  o Constructor objects can be added to the factory
  o Each constructor knows how to instantiate some types

• Why?
  o Allow external libraries to inject custom types without dependencies

```cpp
template<class T>
class AIFactoryBase
{
public:
    T* Create(AICreationData& creationData);

    // Add a custom constructor. Takes ownership of the constructor.
    void AddConstructor(AIConstructorBase<T>* pConstructor);
};
```
Factories - Bells & Whistles

- **Templates & Macros**
  - Consistent naming $\Rightarrow$ automated factory specification

- **Why:**
  - Every factory works exactly the same way
  - Adding a new *type* of object is dead simple
Macro Magic: Declaring Factories

```cpp
#define DECLARE_GAIA_FACTORY(_TypeName) 
    class AI##_TypeName##Base;

    class AI##_TypeName##Constructor_Default : public AICConstructorBase<AI##_TypeName##Base> 
    {
        public:
            virtual AI##_TypeName##Base* Create(const AICreationData& creationData);
    };

    class AI##_TypeName##Factory : public AIFactoryBase<AI##_TypeName##Base> 
    {
        public:
            AI##_TypeName##Factory()
            { AddConstructor(new AI##_TypeName##Constructor_Default); }
    };

#undef DECLARE_GAIA_FACTORY
```
Combining Considerations

- **AIGH! Not enough time!**
  - Kevin Dill (2016): a simple Boolean approach
    - “Quick and Dirty: 2 Lightweight AI Architectures”
  - Mike Lewis & Dave Mark (2015): a utility-based approach
    - “Building a Better Centaur: AI at Massive Scale”
  - Kevin Dill & Dave Mark (2012): a dual-utility approach
    - “Embracing the Dark Art of Mathematical Modeling in AI”

- I strongly recommend the third - it’s:
  - Straightforward to implement
  - Extremely flexible - capable of great power (hardcore utility-based AI) or great simplicity (each consideration is a “yes” or “no”)
  - Avoids combinatoric problems of Mike & Dave’s approach

- You can customize this in the Consideration Set!
Agenda

• What is Modular AI?
• Common Conceptual Abstractions
• Sniper Example
• Implementation
• Parting Thoughts
What Does This Buy Me?

- Appropriate level of abstraction
  - Enter 6 values vs. several hundred lines of code
  - Those values are the relevant ones
- Broad reuse of both components (code) and behavior (XML)
  - Implement once
  - Fewer bugs
  - More mature code (better tested, more feature-rich)
- The Bottom Line: Developer Flow

<Consideration Type="ExecutionHistory">
  <StoppedWeightFunction Type="FloatSequence">
    <Entries>
      <Entry Min="60" Max="90" Veto="true"/>
    </Entries>
    <Default Veto="false"/>
  </StoppedWeightFunction>
</Consideration>
Where To Start?

• You don’t have to build a new architecture from scratch
  o If you do, it doesn’t have to be as complex as mine

• Look for opportunities to build in a modular way
  o Weapon selection
  o Target selection
  o Red Dead example (missed opportunity)

• Start with considerations
The Mars Game

- Simple open-source implementation
  - Apache 2 license
  - GitHub: https://github.com/virtual-world-framework/mars-game
    - or Google “GitHub Mars Game”
Nuts and Bolts: Modular AI from the Ground Up

Kevin Dill
Christopher Dragert
Troy Humphreys
Factories - Bells & Whistles

- Constructors
  - Constructor objects can be added to the factory
  - Each constructor knows how to instantiate some types

- Why?
  - Allow external libraries to inject custom types without dependencies

```cpp
template<class T>
class AIFactoryBase
{
public:
  T* Create(AICreationData& creationData);

  // Add a custom constructor. Takes ownership of the constructor.
  void AddConstructor(AIConstructorBase<T>* pConstructor);
};
```
Major Components

Option
Considerations
• Consideration #1
• Consideration #2
• ...
Actions
• Action #1
• Action #2
• ...

Option
Considerations
• Consideration #1
• Consideration #2
• ...
Actions
• Action #1
• Action #2
• ...

Option
Considerations
• Consideration #1
• Consideration #2
• ...
Actions
• Action #1
• Action #2
• ...

Option
Considerations
• Consideration #1
• Consideration #2
• ...
Actions
• Action #1
• Action #2
• ...
Sniper

- Periodically (every minute or two) takes a shot at the enemy.
  - Not if there is no line of retreat.
  - Not if under fire.
- Withdraws after firing a few shots
- Withdraws if the enemy opens fire
  - If he can’t withdraw, returns fire instead
### Sniper

**Rule-Based Reasoner**

#### Withdraw Considerations
- **Blackboard** (# Shots Fired)
- **Event** (Under Fire)
- **Picker** (Line of Retreat)
- **Execution History** (Commit)

**Actions**
- **Withdraw**

#### Fight Considerations
- **Event** (Under Fire)
- **Picker** (Select Target)
- **Execution History** (Commit)

**Actions**
- **Fire at Target**

#### Take A Shot Considerations
- **Execution History** (Timer)
- **Picker** (Select Target)
- **Picker** (Line of Retreat)

**Actions**
- **Fire at Target**
- **Write Blackboard** (# Shots Fired)

#### Hide Considerations
- **Pose** (Low Prone)

**Considerations**
- Periodically (every minute or two) takes a shot at the enemy.
  - Doesn’t fire if there is no line of retreat.
  - Doesn’t fire if under fire.
- Withdraws after firing a few shots
- Withdraws if the enemy opens fire
  - If he can’t withdraw, returns fire instead
Hierarchy

Rule-Based Reasoner

- Periodically (every minute or two) takes a shot at the enemy.
  - Doesn’t fire if there is no line of retreat.
  - Doesn’t fire if under fire.
- Withdraws after firing a few shots
- Withdraws if the enemy opens fire
  - If he can’t withdraw, returns fire instead

Dual Utility Reasoner

- Periodically (every minute or two) takes a shot at the enemy.
  - Doesn’t fire if there is no line of retreat.
  - Doesn’t fire if under fire.

Actions

- Raise Up
  - Pose (High Prone)
- Aim
  - Delay
- Fire
  - Fire Weapon (Target)

Withdraw

Fight

Take a Shot

Hide
Sniper - The “Hide” Option

Hide
Considerations

Actions
• Pose (Low Prone)

```xml
<Option Type="ConsiderationAndAction" Comment="Hide">
  <Considerations/>
  <Actions>
    <Action Type="Pose" Pose="LowProne"/>
  </Actions>
</Option>
```
```
<Configuration>
  <Name></Name>
  <Reasoners>
    <Reasoner><Type X></Reasoner>*
  </Reasoners>
  <Options>
    <Option><Type X></Option>*
  </Options>
  <Considerations>
    <Consideration><Type X></Consideration>*
  </Considerations>
  <Actions>
    <Action><Type X></Action>*
  </Actions>
</Configuration>

AIConfiguration (loaded by AlBrain)

AIReasonerSet

AIReasonerBase

AlOptionFactory

AlOptionBase

AlConsiderationFactory

AlConsiderationBase

AlActionFactory

AlActionBase

AlActionSet
```
Tools

Scenario Behavior Editor

Alpha (January 2016)

Start State (Actions)
- Action
  - Name: missionStart
    - Brief: This is a quick summary.
    - Next Scenario: missionTasks
  - (1 Argument)
  - Arguments
    - startingMusic

Trigger
- Name: baseScenarioDefaults
- Clause: OnScenarioStart
- Actions
  - Action
    - Name: ???
    - Arguments

YAML definition

```
# Copyright 2016 Lockheed Martin Corporation
#
# Licensed under the Apache License, Version 2.0 (the "License"); you may
# not use this file except in compliance with the License. You may obtain
# a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# YAML version 1.2
extends: ...
scenePath: ...
scenarioName: "missionTask0"
startState:
  - playSound:
    - startingMusic
  - setProperty:
    - rover
children:
  triggerManager:
    extends: ...
    properties:
      triggers:
      - reactScenarioDefaults:
        - triggerCondition:
```