AI ARTIFICIAL INTELLIGENCE SUMMIT

Nuts and Bolts: Modular AI from the Ground Up

Kevin Dill
Christopher Dragert
Troy Humphreys

Abstract

What is **Modular AI**?

- It's a way to structure your **AI Architecture**
  - Applies to state machines, behavior trees, HTNs, etc.
- Emphasises small, easily **reused** modules
- Can be **transformative** to your development process
  - Fast prototyping, rapid iteration, increased stability

Abstract

The Nuts and Bolts

1. Academic Underpinnings (Chris Dragert)
2. Implementation Details with Code Samples (Kevin Dill)
3. Shipped Example and Architecture Discussion (Troy Humphreys)

ARTIFICIAL INTELLIGENCE SUMMIT

Nuts and Bolts: Modular AI from the Ground Up

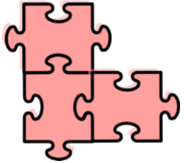
Christopher Dragert, Ph.D.
Ubisoft Toronto

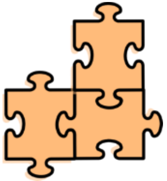
Introduction

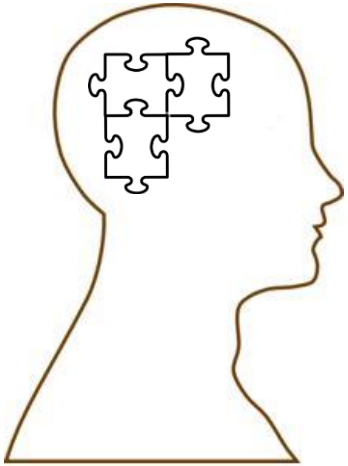
Modular AI

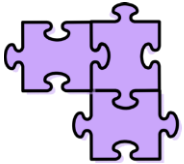
- Software engineering has a lot to say about **modular reuse**

- Apply these principles to **modular AI**









Introduction

Our Goals

- Learn techniques to develop a suitable **modularization** for your project
- Understand how to manage and reduce **modular complexity**

Complexity

Classifying Complexity

- **Essential complexity**

  - Complexity of the problem itself

- **Accidental complexity**

  - Problems created by us

[Fred Brooks, "No Silver Bullet", 1986]

Complexity

What drives Modular Complexity?

1. The **Module** itself
2. Complexity of the **Interface**
3. The **Integration** process

Module Complexity

Module Complexity

- Good modules do not try to do too much!
- Smaller modules improve **comprehension** by having **singular purpose**

Module Complexity

Limiting Scope

- Separate **cross-cutting concerns**
  - *Example* - Melee combat module selects a target, ranged module selects a target, flee module selects a target…
  - *Solution* - Remove target selection from existing modules, create a **target selection module**

Module Complexity

Control the Size

- Traditional abstraction techniques should be applied
  - Hierarchical Approaches
  - Subsumption and Layering
  - Parallelism

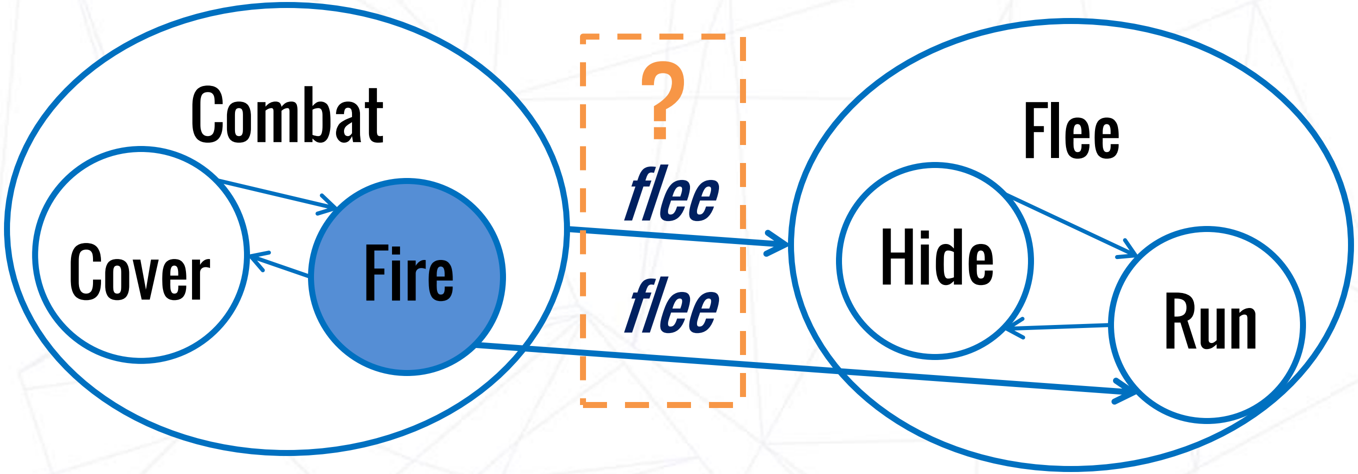
Module Complexity

Well-Defined Semantics

- Your AI logic must operate in a understandable, **well-defined** fashion
- Necessary for **portability** between games

Module Complexity

Semantics Example



- What transition does your implementation take?
  - The new context must make the same choice!

Interface Complexity

Modular Interface

- Communicates the required **context** for the module
- Raises the level of **abstraction**, reducing accidental complexity

Interface Complexity

Defining the Context

- State machines (event-based formalisms)
    - What **input events** in do you need to handle?
    - What **output events** do you generate?

Interface Complexity

Enemy Position Tracker

Interface Complexity

Enemy Position Tracker

Description: Tracks the position of an enemy
Game: ‘Game X’ by Ubisoft
Parameters: <T> The type of the enemy entity
Language: C++

Interface Complexity

Enemy Position Tracker

Description: Tracks the position of an enemy
Game: ‘Game X’ by Ubisoft
Parameters: <T> The type of the enemy entity
Language: C++

Input Events

```
- ev_EnemySpotted(<T> enemy)
- ev_EnemyLost(<T> enemy)
```

Interface Complexity

Enemy Position Tracker

Description: Tracks the position of an enemy
Game: 'Game X' by Ubisoft
Parameters: <T> The type of the enemy entity
Language: C++

Input Events	Output Events

`- ev_EnemySpotted(<T> enemy)`
`- ev_EnemyLost(<T> enemy)`
	`-ev_EnemyPositionChanged (<T> enemy)`

Interface Complexity

Behavior Tree Contexts

- Primarily data-driven
  - What **blackboard entries** are read (input) and written (output)?
- Not the full story!

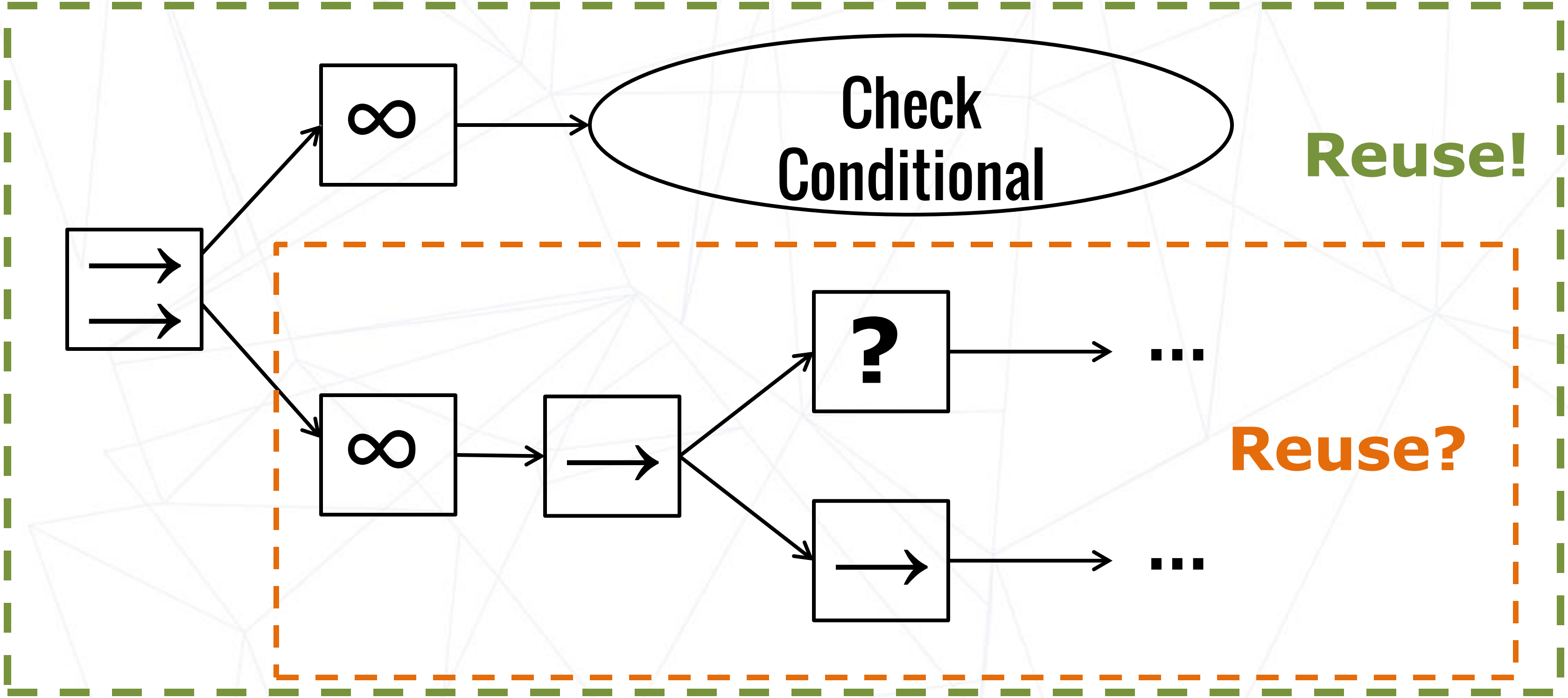
Interface Complexity

Behavior Tree Contexts

- New behavior trees where nodes can return {success, failure, running}
  - What **interrupts** a running node?
- Tree structure itself

Interface Complexity

Behavior Tree Interfaces



Integration Complexity

Integration Overview

- The essential problem is **connecting** inputs and outputs between modules
- Everything else is **accidental complexity**!

Integration Complexity

Integration Complexity

- Module **connections** must be derivable solely from the interface
    - This preserves **modular encapsulation**!

- A consistent integration approach can be supported with **tools**

Integration Complexity

Module Coupling

- **Loosely-Coupled**: A missing module impairs only that behavior

- Loosely coupled modules support **fast prototyping** and **rapid iteration**

Integration Complexity

Module Coupling

- **Tightly-Coupled**: Missing modules cause failures, prevent compilation, etc
  - Often caused by **broken encapsulation**
  - Could also be an error in abstracting modular concerns

Integration Complexity

Special Cases

- Special case exceptions break reuse
  - **Sensor**: Reports every `ev_newEnemySpotted` event
  - **Reaction**: `ev_newEnemySpotted` causes a new enemy reaction
  - Event system adds hysteresis, caps generation of `ev_newEnemySpotted` at one per minute
- This is a broken module encapsulation error

Summary

The Payoff

- **Fast Prototyping**
  - Quickly modify functionality by adding and removing modules
- **Fine Tuning**
  - Parameterized module instances allow for customization
- **Better Development Process**
  - Reuse existing behavior, spend time innovating new behaviors

Summary

A good modular approach:

- Uses **small modules** that separate concerns
- Operates with **well-defined semantics**
- Has a **clear interface**
- Preserves modular **encapsulation**
- Uses a **loose-coupling** approach