Sean Middleditch

Who I Am

DigiPen RTIS Senior

Subsonic, SONAR, Core

http://seanmiddleditch.com

Introduction

Just what are Action Lists?

You've Seen Them Already

"Action" ... "List"

They're lists...

... of actions!

Not Really a List

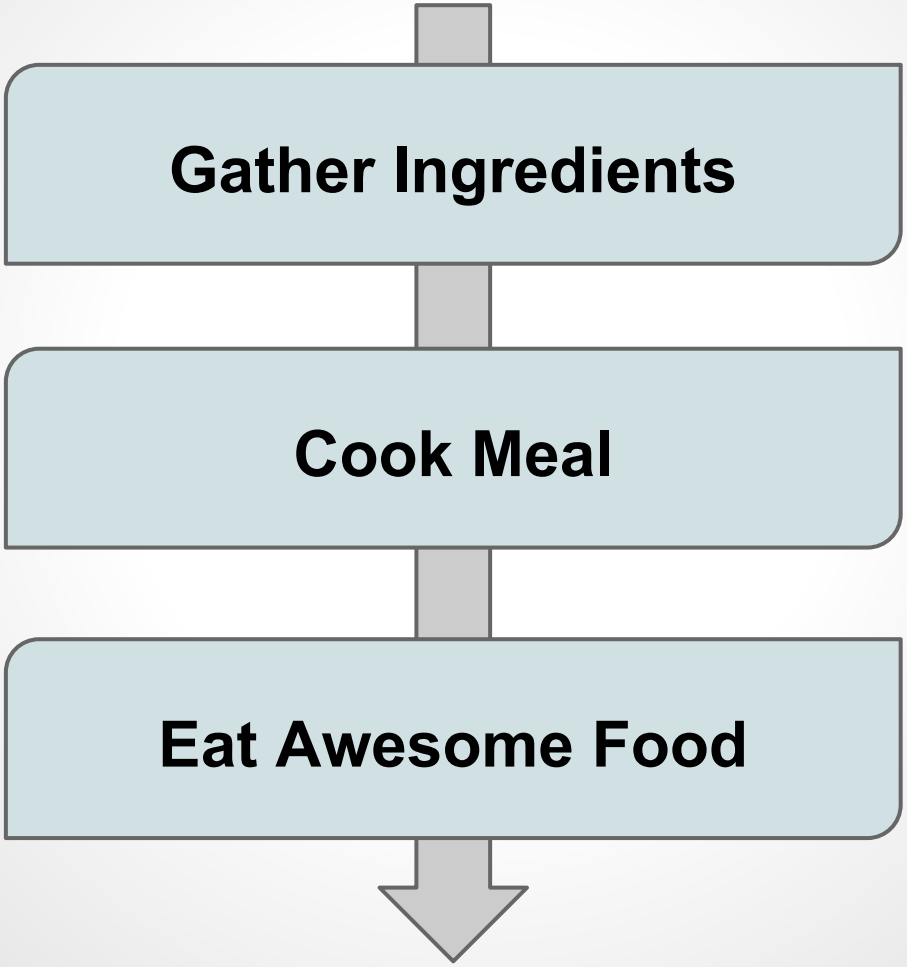
Basic action "lists" are actually queues (FIFO)

They contain an ordered list of actions

Each is called in sequence

Do X, then do Y, then do Z

Action ~~Queue~~ Lists



Executing Actions

Actions are executable

They can take more than one frame to complete

Action::Update() method is called every frame until the action indicates that it is complete

Only the first action is updated

Simple Action Interface

```
// base action interface
class Action
{
public:
    Action() : IsFinished(false) {}
    virtual ~Action() {}

    virtual void Update() = 0;

    bool IsFinished;
};
```

Update() sets IsFinished to true when complete

Simple WalkTo Action

```
// walk to a point
class WalkTo
{
public:
    WalkTo(const vec3& position, float speed) :
        Position(position), Speed(speed) {}

    virtual void Update()
    {
        if (AreWeThereYet(Position))
            IsFinished = true;
        else
            MoveTowards(Position, Speed);
    }

    vec3 Position;
    float Speed;
};
```

Simple ActionList Loop

```
// in header
class ActionList
{
public:
    void Update();
    void PushAction(Action* action) { Actions.push(action); }

    std::queue<Action*> Actions;
};

// in code file
void ActionList::Update()
{
    if (!Actions.empty())
    {
        Actions.front()->Update();
        if (Actions.front()->IsFinished)
            Actions.pop();
    }
}
```

Parallel Actions

Can he walk and wave at the same time?

Blocking vs Non-Blocking

a.k.a. Synchronous or Asynchronous

Blocking means that one action stops the second from running

Non-blocking means that the first action allows the second to run simultaneously

Mixing Blocking & Non-Blocking

The standard action list has only blocking actions

Allowing a mix of blocking and non-blocking gives a lot of flexibility

Complex behaviors can be made by combining blocking and non-blocking actions

Simple Blocking Action Interface

```
// base action interface
class Action
{
public:
    Action(bool blocking) : IsBlocking(blocking), IsFinished(false) {}
    virtual ~Action() {}

    virtual void OnUpdate() = 0;

    bool IsBlocking;
    bool IsFinished;
};
```

Action can be configured as Blocking or Non-Blocking at runtime

Simple Blocking WalkTo Action

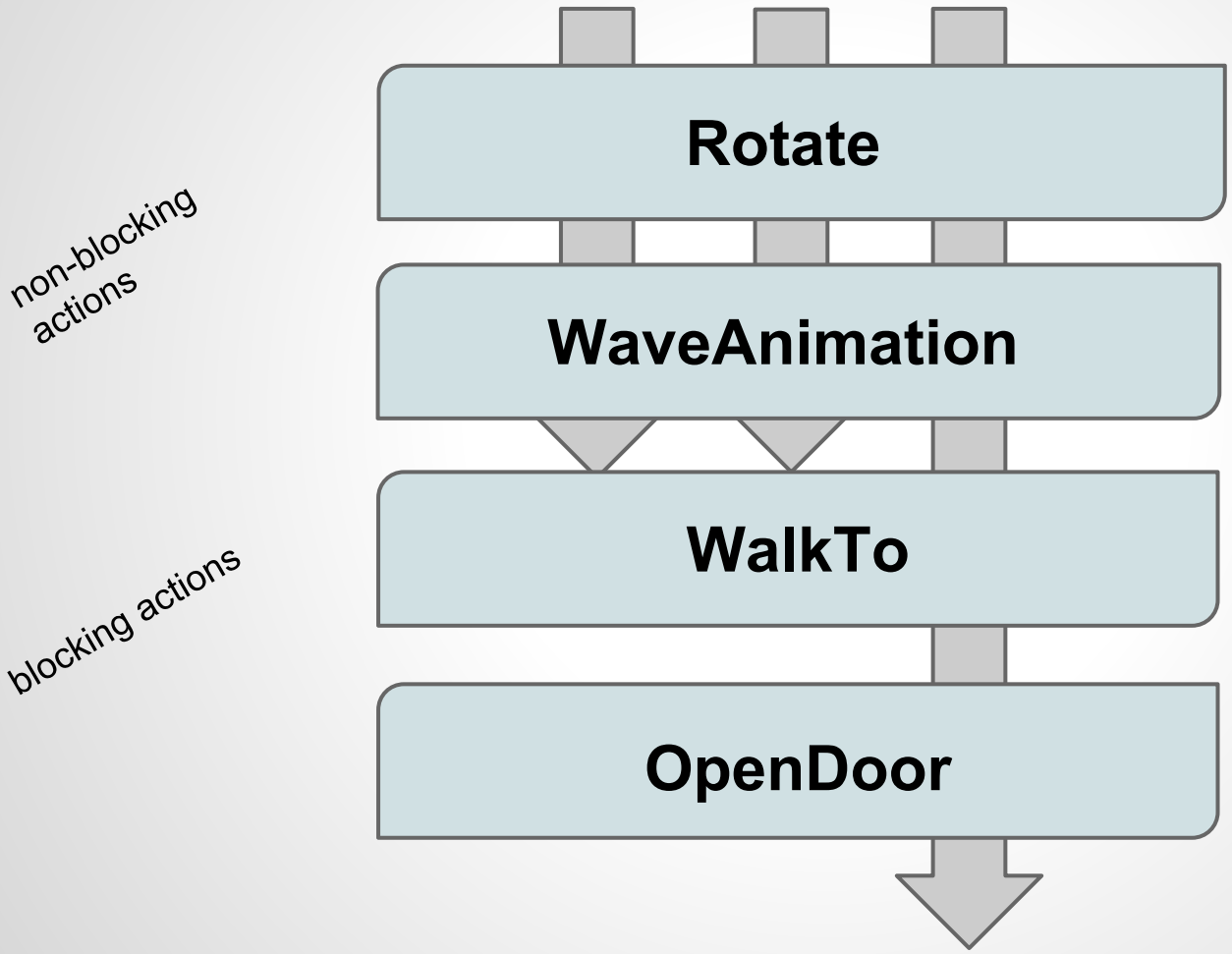
```
// walk to a point
class WalkTo
{
public:
    WalkTo(bool blocking, const vec3& position, float speed) :
        Action(blocking), Position(position), Speed(speed) {}

    virtual void OnUpdate()
    {
        if (AreWeThereYet(Position))
            IsFinished = true;
        else
            MoveTowards(Position, Speed);
    }

    vec3 Position;
    float Speed;
};
```

Simple Blocking ActionList Loop

```
std::vector<Action*>::iterator iter = m_Actions.begin();
bool blocked = false;

while (!blocked && iter != m_Actions.end())
{
    Action* action = *iter;

    action->OnUpdate();

    blocked = action->IsBlocking;

    if (action->IsFinished)
        iter = m_Actions.erase(iter);
    else
        ++iter;
}
```

Execute each action until a blocking action runs, remove completed actions from your action list

Visualizing as Parallel Queues



Sync Action

What if you want three non-blocking actions to run, and then one blocking action?

Create a Sync action that is Blocking

It sets IsFinished to true when it is the first action in the list

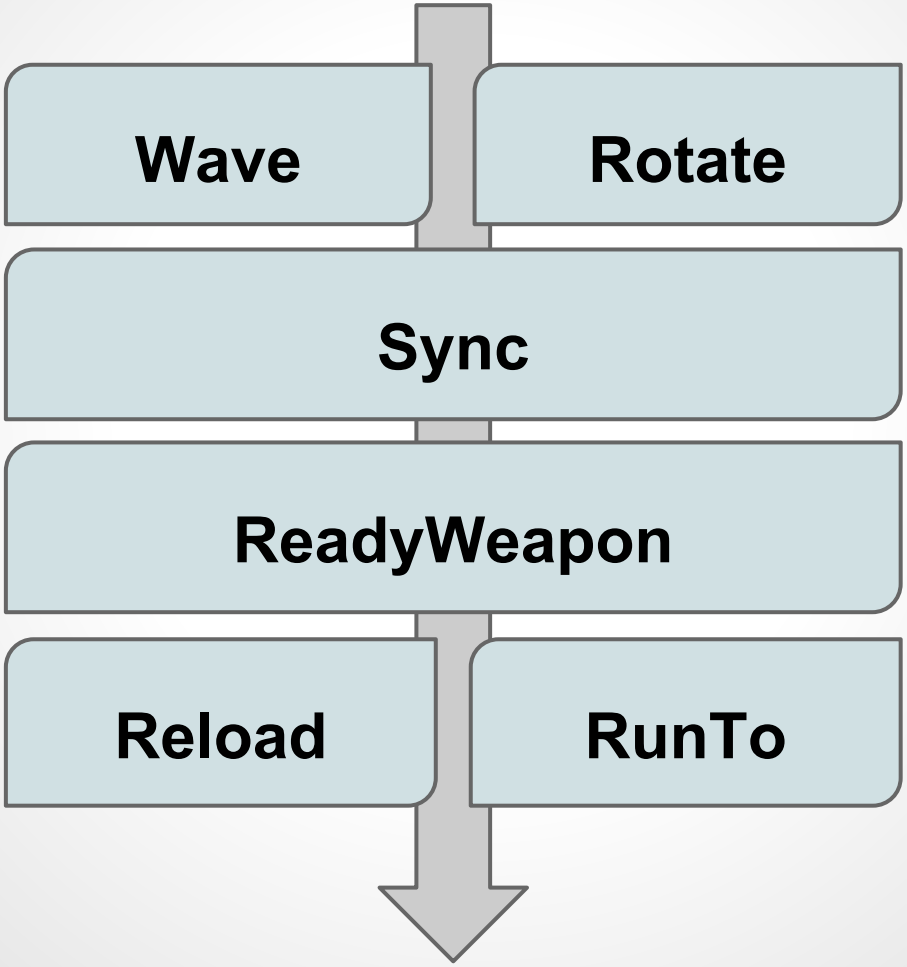
Simple Blocking WalkTo Action

```
// sync action
class Sync : public Action
{
public:
    virtual void Update()
    {
        if (OwningActionList.Actions.front() == this)
            IsFinished = true;
    }
};
```

Good time to note: use better abstraction, private members, accessor functions, name members with m_ prefix, etc.

Slides are just saving space 

Sync Action Diagram



Other Considerations

You might give Action a built-in delay

Or a built-in time to finish

Utility functions like IsEmpty(), IsFirst(), etc.

DidRunAlready flag

Lanes

a.k.a. bitmasks

Improving on Simple Blocking

Simple blocking vs non-blocking is limited

Sync action helps, but still limited

Difficult to combine actions in interesting ways

Lanes to the Rescue

Actions are assigned to one or more Lanes

Only one Action in any Lane can ever run

If any Lane an Action is in has been blocked, the Action is blocked

Actions run in order as usual, but blocked Actions may be between two unblocked actions

Selective Blocking via Bitmasks

Each Action has a bitmask, termed Lanes

The ActionList keeps a cummulative bitmask during each update loop (bitwise OR the lanes of the executed Actions)

If an Action's Lanes bitmask intersets the cummulative bitmask (bitwise AND is non-zero), the Action is skipped/blocked

Simple Blocking WalkTo Action

```
class Action
{
public:
    Action(bool blocking, int lanes) : Lanes(lanes), IsBlocking(blocking),
        IsFinished(false) {}

    virtual void Update() = 0;

    int Lanes;
    bool IsBlocking;
    bool IsFinished;
}
```

Setting Lanes in constructor makes it easy for derived classes to configure defaults, or allow users to override the lanes on a per-action basis

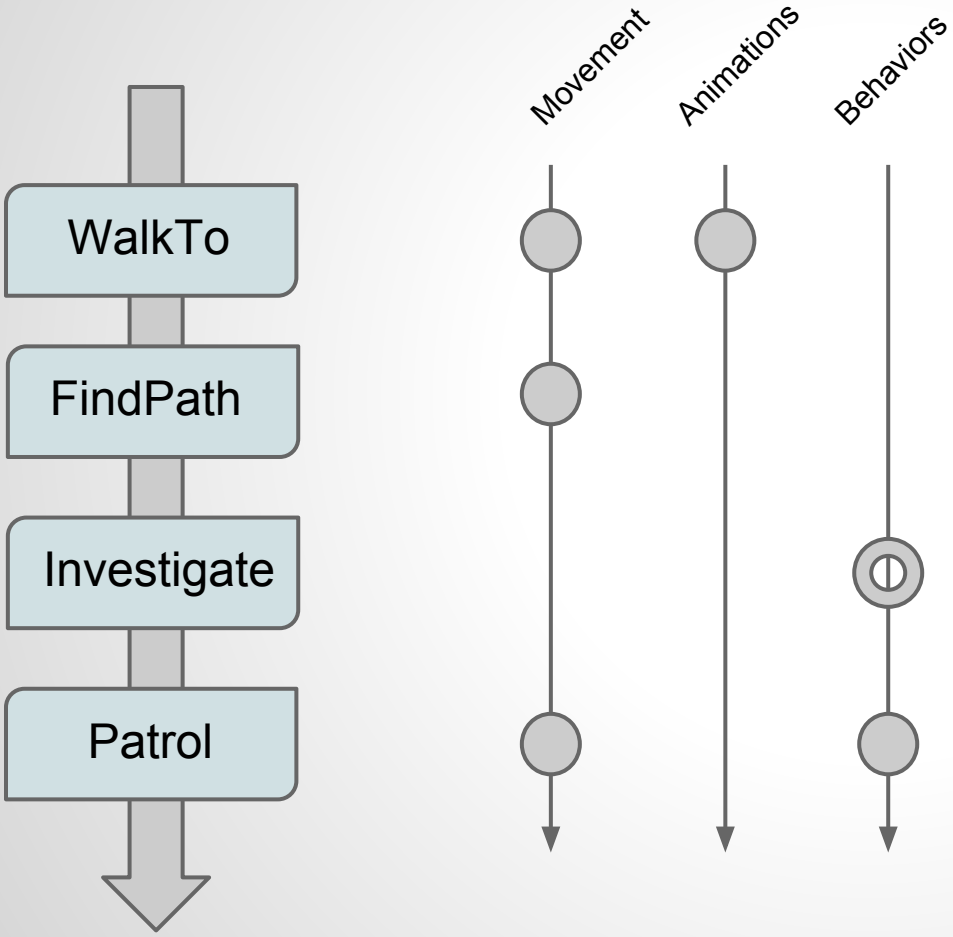
Simple Blocking WalkTo Action

```
std::vector<Action*>::iterator iter = m_Actions.begin();
int mask = 0;

while (iter != m_Actions.end())
{
    Action* action = *iter;

    if (0 == (mask & action->Lanes))
    {
        if (action->IsBlocking)
            mask |= action->Lanes;

        action->Update();

        if (action->IsFinished)
            iter = m_Actions.erase(iter);
        else
            ++iter;
    }
    else
        ++iter;
}
```

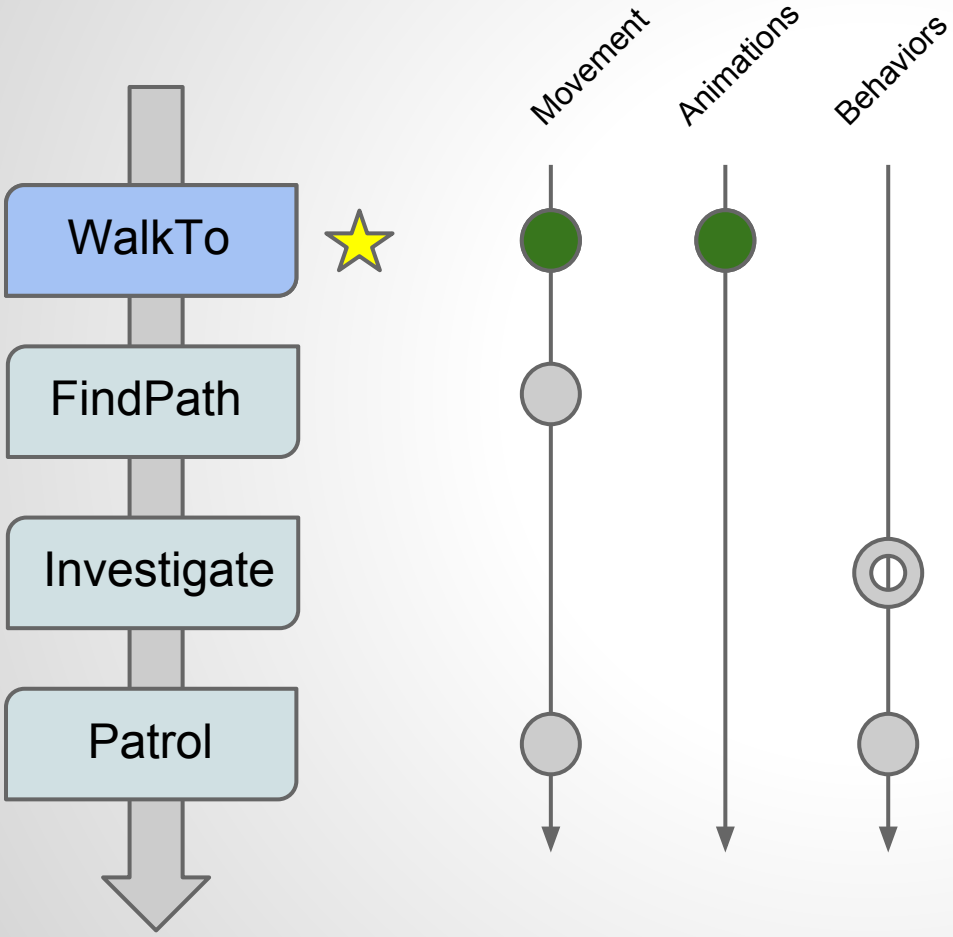
Example with Lanes

Can bad graphics make this clearer?

Example with Lanes

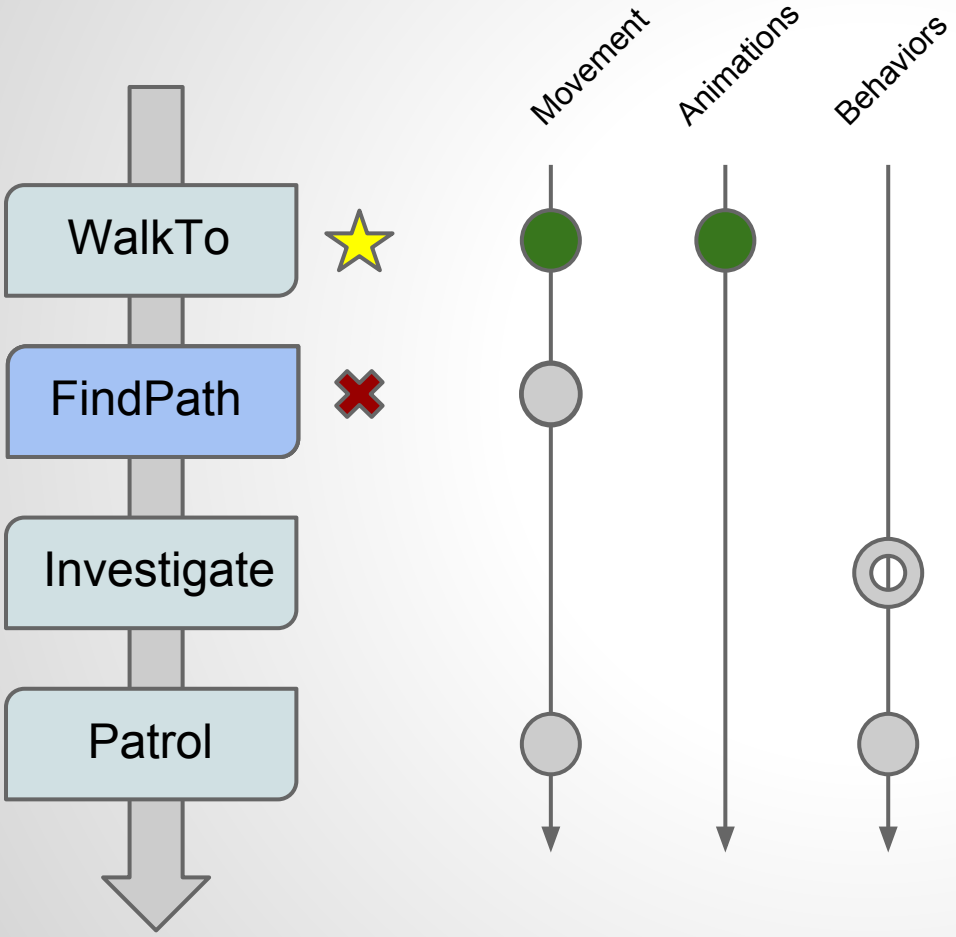


Example with Lanes



WalkTo blocks all movement and animations

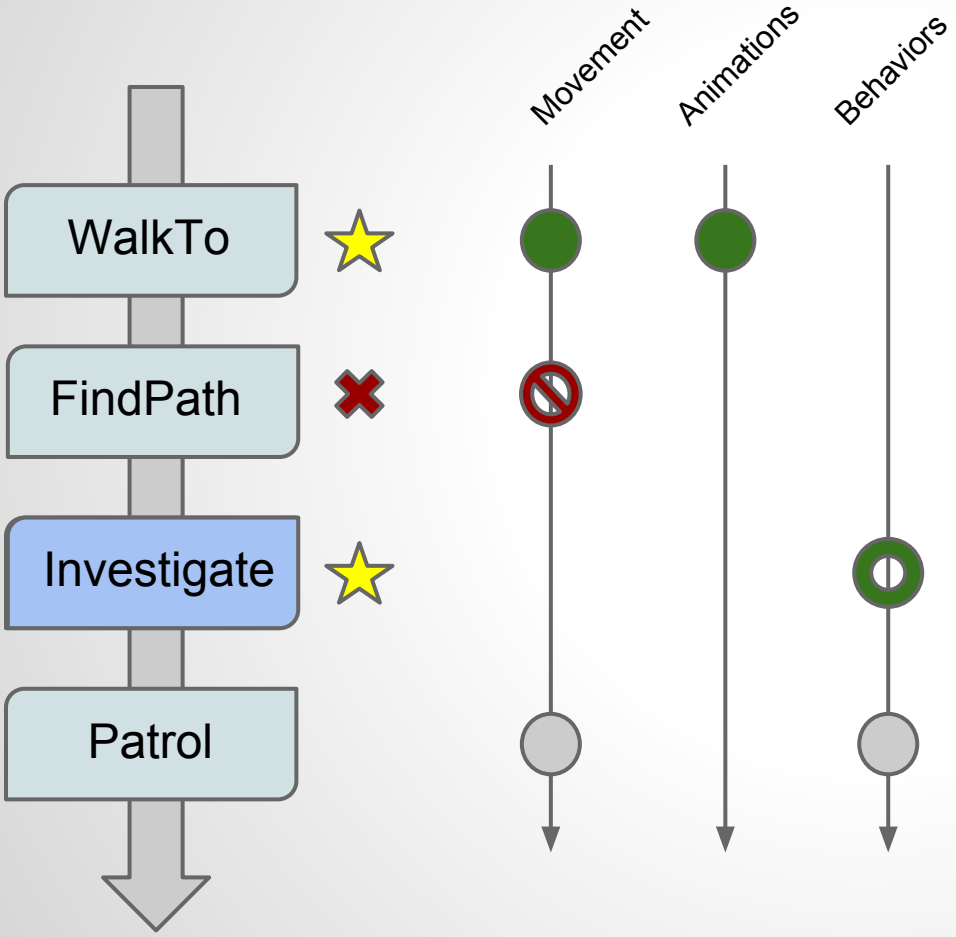
Example with Lanes



WalkTo blocks all movement and animations

FindPath is blocked

Example with Lanes

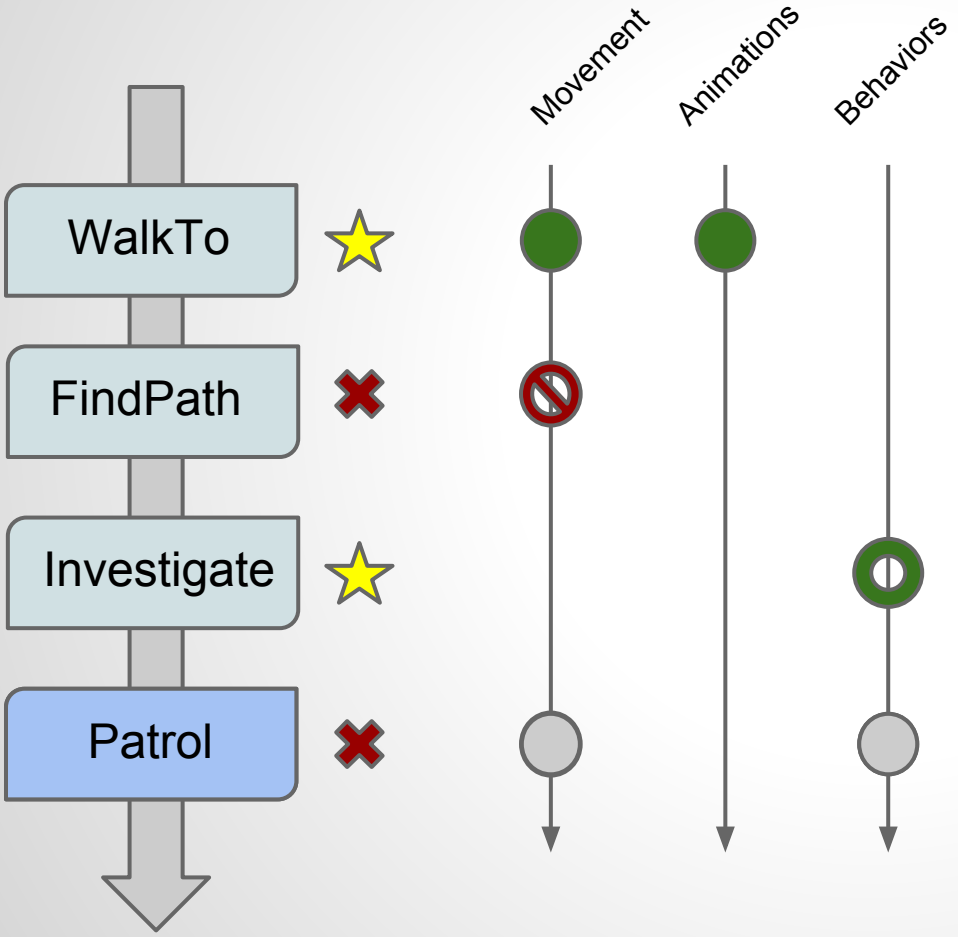


WalkTo blocks all movement and animations

FindPath is blocked

Investigate runs, but isn't blocking

Example with Lanes



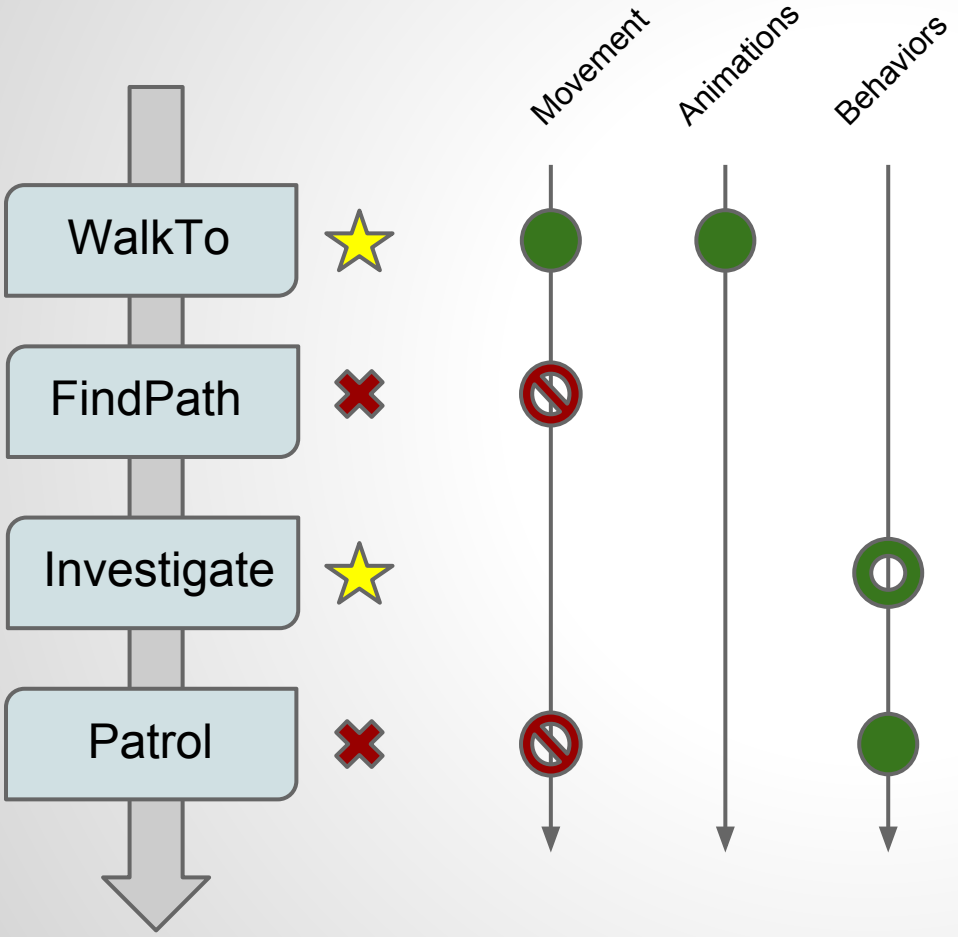
WalkTo blocks all movement and animations

FindPath is blocked by WalkTo

Investigate runs, but isn't blocking

Patrol is blocked by WalkTo

Example with Lanes



WalkTo blocks all movement and animations

FindPath is blocked by WalkTo

Investigate runs, but isn't blocking

Patrol is blocked by WalkTo

Improving on Lanes

Two separate bitmasks

- One for "what I will block"
- Another for "what I am blocked by"

Hierarchical Action! Lists

Yo dawg, I heard you like action lists...

Feudal Action!

Each game object has a primary ActionList

Some Actions may have their own complex behavior and contain an ActionList

The embedded ActionLists are blocked if their parent Action is blocked, and run otherwise

Simple Blocking WalkTo Action

```
class HierarchicalAction : public Action
{
public:
    virtual void Update()
    {
        MyActionList.Update();
        IsFinished = MyActionList.IsEmpty();
    }

    ActionList MyActionList;
};
```

Naturally, the HierarchicalAction could have its own specialized logic in addition to the embedded ActionList

A Few More Suggestions

Miscellania that may or may not be useful

Simplified Lanes

Use a single Lane per Action

Just a number

Calculate bitmask using shifts

```
int mask = 1 << action->Lane;
```

Parent/Child Relationships

Hierarchical Actions in one ActionList

When an Action spawns more Actions (e.g., PathTo generates a series of MoveTo's), the child Action gets a pointer back to parent

Allows removing an Action and all its children at once, useful for complex cancellable Actions

Breaking the FIFO Mold

One of the more useful features

Allow pushing Actions to the front or back of the list

Allow inserting Action before or after another Action

Block and Unblock Events

Add Blocked() and Unblocked() methods

Some Actions may have special behavior to suspend or resume when they get blocked

Especially useful when combined with the previous suggestion

Paused Event

Allow setting an Acton to a paused state

It does not run, and does not block anything else

Can have Paused()/Unpaused() events too

Combine with Blocked() event and Parent/Child Actions for complex systems (parent is blocked, so all children get paused)

Message Delivery

Super important feature for most usecases

Actions need to know what's going

Let them observe messages, or deliver any messages delivered to a game object to its ActionList as well

OnMessage() method or similar

Debugging Action Lists

Managing the complexity

Debugging

Every Action has state imposed by the system:

Is blocked, is blocking, is paused, lanes, parent, position in list, prev/next actions, etc.

Each Action has its own specialized state:

WalkTo's target position, PlayAnimation's animation, etc.

Debugging

Actions have derived state:

Who is blocking me, who am I blocking, who are my children, etc.

Each Action can have optional debugging state:

Time Action has been running

Visualization

#1 WalkTo (0x8030) {blocking, #2, #3} position=4.5,1.7

#2 WalkTo (0x8030) [blocked, #1] {blocking} position=7.2,2.1

#3 PickUp (0x2030) [blocked, #1] object=Longsword

Even with ugly text-only output, this can be immensely helpful in debugging what an ActionList is doing and what the problem is

Behavioral Composition 1.0

Using action lists for AI behavior

Action Lists in AI

Often used for managing AI-chosen actions

Rarely used for managing the AI itself

AI decides to path to a location, pushes the WalkTo (or PathTo) actions to the ActionList

AI Architecture

AI is often implemented as FSM (Finite State Machine), which are inflexible, difficult to write, difficult to edit for designers, and difficult to debug

Advanced decision making AI architectures (Goal Planning, Behavior Trees, etc.) are very flexible, but still difficult to write, difficult to edit for designers, and difficult to debug

AI Wishlist

1. Minimal amount of architecture to write
2. Easy to build new AI (composable)
3. Good debugging tools
4. Flexible enough for our needs

AI ~~Wishlist~~ in Action Lists

1. ActionList already written
2. Compose individual Behavior Actions
3. ActionList is easily debuggable
4. Flexibility with Lanes, Parent/Child, Blocked/Unblocked/Paused/Unpaused Events

Behaviors

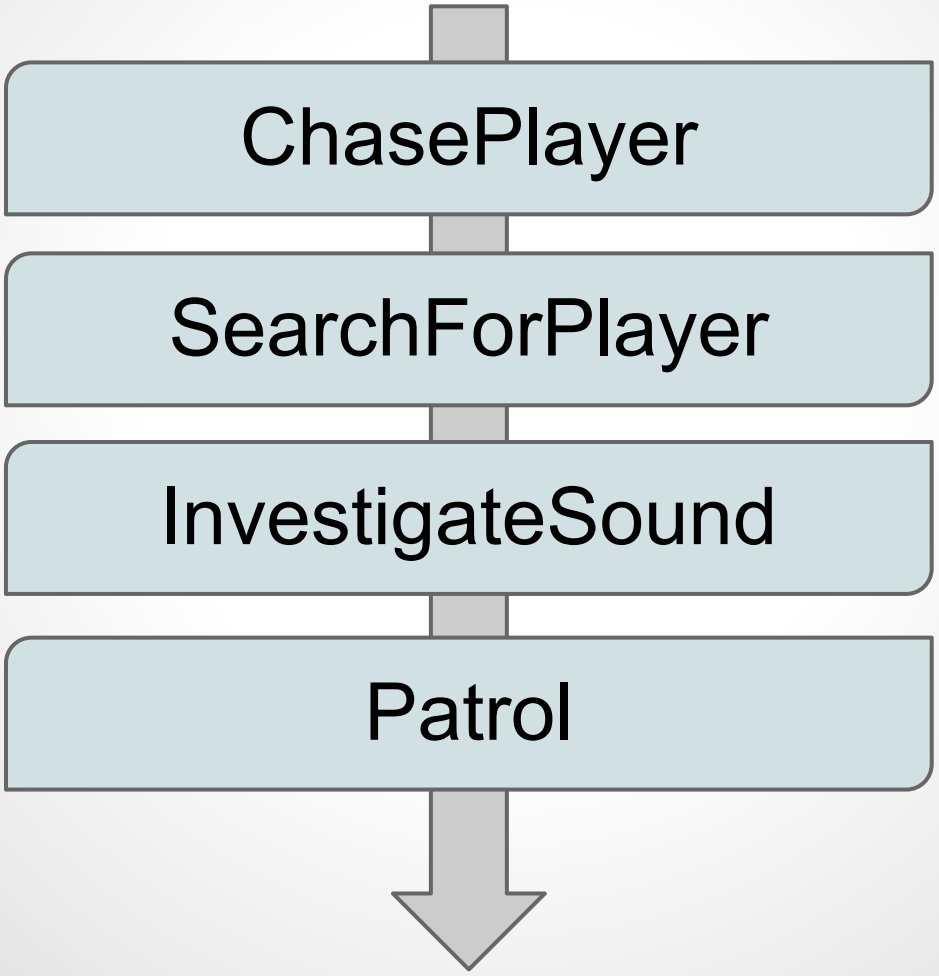
Behaviors are just Actions

Behaviors are never finished

Active Behaviors are blocking

Prioritized by importance

Simple Behavior List



Simple Behavior List Part Deux

ChasePlayer is active when player is visible

SearchForPlayer is active when player is lost

InvestigateSound is active when hearing noise

Patrol is always active

Simple Behavior List Part Trois

If the player becomes visible, ChasePlayer becomes active and sets its IsBlocking flag to true

No other Behaviors further down the list run while blocked

When the player is lost, ChasePlayer deactivates, but SearchForPlayer activates

Simple Behavior List Part Quatre

SearchForPlayer deactivates on its own after several seconds

Patrol is always active, but it's at the bottom of the list, so it only runs when nothing else is going on

New variations of the enemy can be created by adding, removing, or reordering Behaviors

More Action List Uses

They're like duct tape

Graphics

Action lists can be used for post-processing chains

They can be used for animation queues

UI elements can be animated, sequences like "shake screen" or "fade to black" can be in a queue, etc.

Game State Manager

Each game state is an Action

ActionList is used to manage states

Can push/pop states like PauseScreen or OptionsScreen

Use messages, Blocked/Unblocked, Paused/Unpaused states to allow overlays

Scripting...?

Very poor man's scripting for traps or puzzles

Inflexible but very easy to edit

Each Action waits for a specific trigger to unblock itself, final Action opens door or pit trap

Le Questions