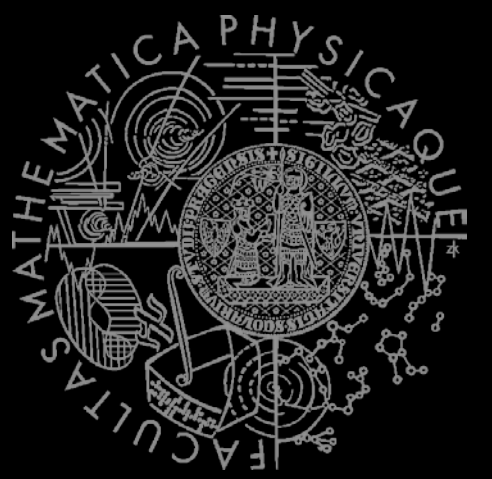
Faculty of Mathematics and Physics
Charles University in Prague
29th March 2016



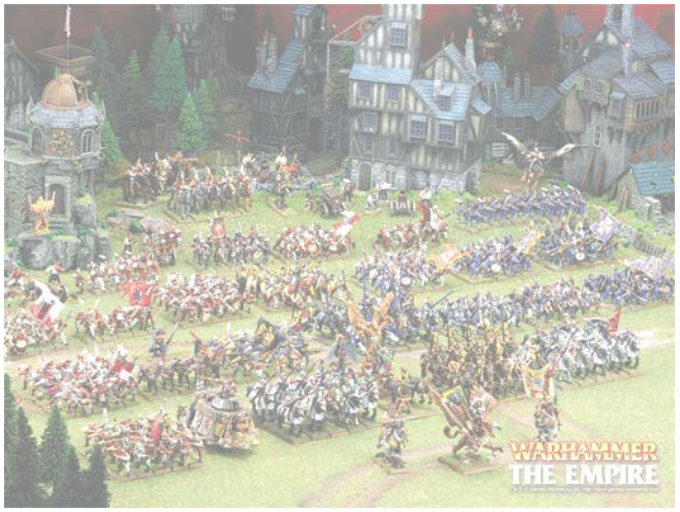
Human-like Artifical Agents

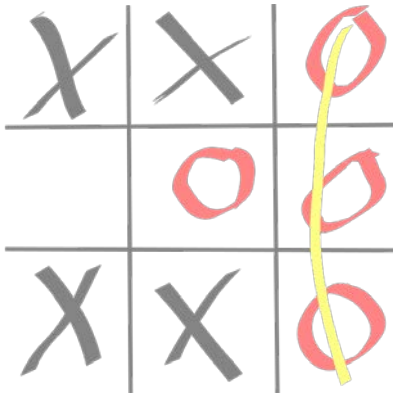
Reactive Planning – Part II

Scripting Virtual Brain

3D V-Environments
What can be said?

- Fully *vs.* Partially observable
- Episodic *vs.* Sequential
- Static *vs.* Dynamic
- Single *vs.* Multi agent
- Deterministic *vs.* Stochastic
- Discrete *vs.* Continuous
- Known *vs.* Unknown
- Turn-based *vs.* Real-time
- Noiseless *vs.* Noisy

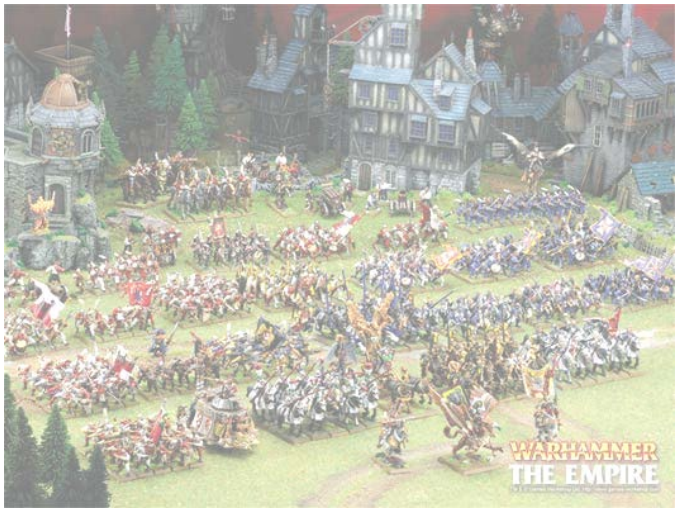


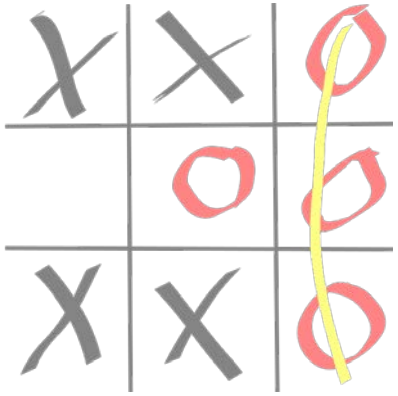


3D V-Environments

Hard to "search or plan"

- Fully *vs.* Partially observable
- Episodic *vs.* Sequential
- Static *vs.* Dynamic
- Single *vs.* Multi agent
- Deterministic *vs.* Stochastic (weakly)
- Discrete *vs.* Continuous
- Known *vs.* Unknown (weakly)
- Turn-based *vs.* Real-time
- Noiseless *vs.* Noisy

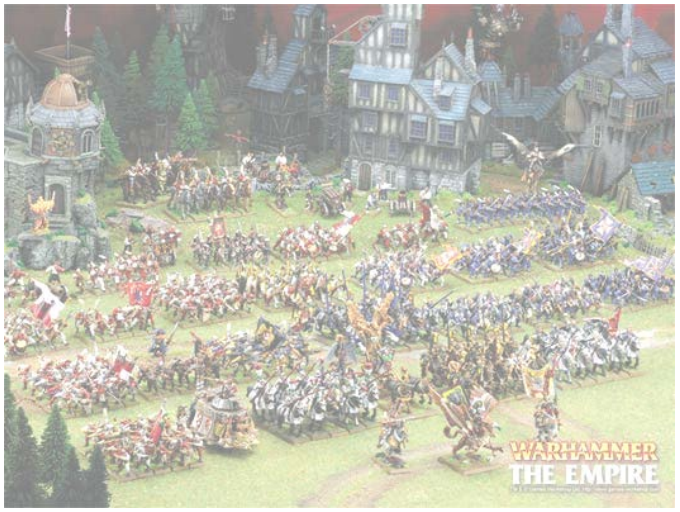


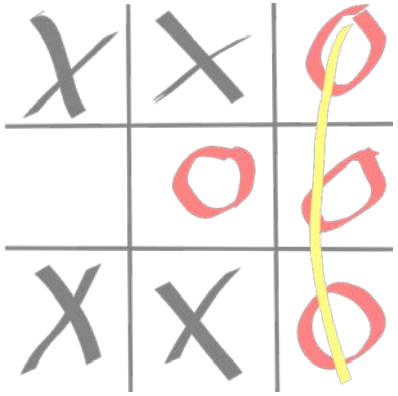


3D V-Environments

=> (Semi) Reactive Action-Selection

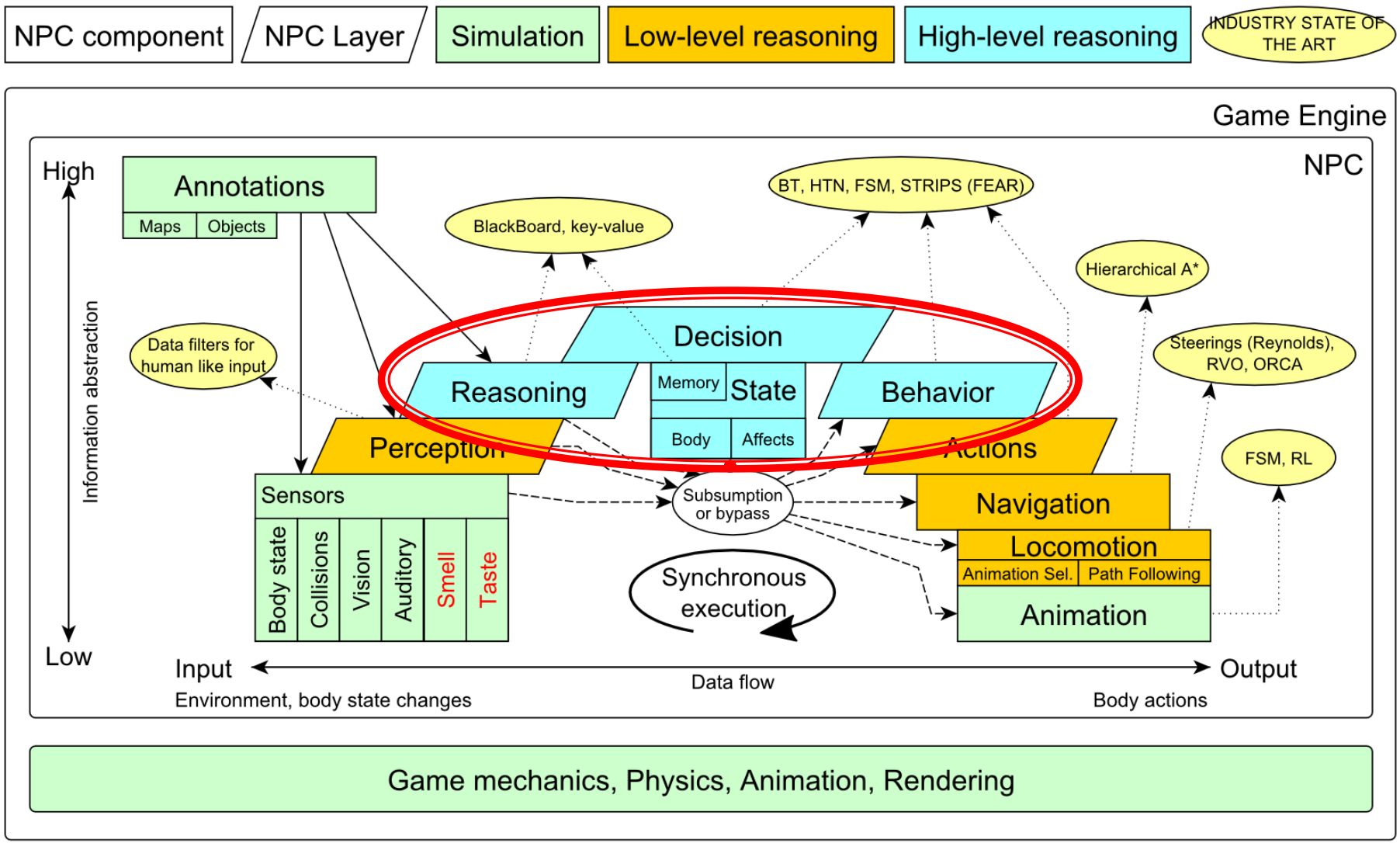
- Fully *vs.* Partially observable
- Episodic *vs.* Sequential
- Static *vs.* Dynamic
- Single *vs.* Multi agent
- Deterministic *vs.* Stochastic (weakly)
- Discrete *vs.* Continuous
- Known *vs.* Unknown (weakly)
- Turn-based *vs.* Real-time
- Noiseless *vs.* Noisy





IVA in Video Games

"Action-selection"



Reactive Planning

Can't we just script everything?

- Programming languages are Turing-complete
  => There is "nothing" you could not do!
- Yeah, but would you use Assembler to write an event-driven GUI application?

  Now I want to create button of different shape, but it should retain all existing features of the button I already have … hmm … with no OOP features though!
- Would you use Java or C# to mimic SQL queries?

  Now I want to left-join table A and B using this condition and group results by column C!

  But nevermind - let's try it!

Procedural Scripting

UT2004 FPS Bot Example

```
public void logic() throws PogamutException {
    if (    weaponry.hasLoadedWeapon(UT2004ItemType.LIGHTNING_GUN)
        || weaponry.hasLoadedWeapon(UT2004ItemType.MINIGUN)) {
        if      (weaponry.hasLoadedWeapon(UT2004ItemType.LIGHTNING_GUN)) weaponry.changeWeapon(UT2004ItemType.LIGHTNING_GUN);
        else if (weaponry.hasLoadedWeapon(UT2004ItemType.MINIGUN))       weaponry.changeWeapon(UT2004ItemType.MINIGUN);
    }
    Player player = players.getNearestVisiblePlayer();
    if (player != null) {
        shoot.shoot(player);
        if (weaponry.hasLoadedWeapon(UT2004ItemType.LIGHTNING_GUN) || weaponry.hasLoadedWeapon(UT2004ItemType.MINIGUN)) {
            navigation.navigate(player);
            return;
        }
    }
    if (info.getHealth() < 50) {
        Item health = fwMap.getNearestItem(items.getSpawnedItems(UT2004ItemType.HEALTH_PACK).values(),
                                           info.getNearestNavPoint());
        if (health != null) {
            navigation.navigate(health);
            return;
        }
    }
    if (navigation.isNavigating()) return;
    Item weapon = null;
    for (ItemType weaponType : ItemType.Category.WEAPON.getTypes()) {
        if (weaponry.hasWeapon(weaponType)) continue;
        weapon = fwMap.getNearestItem(items.getAllItems(weaponType).values(), info.getNearestNavPoint());
        if (weapon != null) break;
    }
    if (weapon != null) {
        navigation.navigate(weapon.getNavPoint());
        return;
    }
    navigation.navigate(navPoints.getRandomNavPoint());
}
```

Behavior Oriented Design *(by J.J.Bryson)*
Agent Behavior Development Methodology

" BOD is a methodology for developing control of complex intelligent agents, such as virtual reality characters, … "

-- J.J.Bryson, University of Bath, UK

http://www.cs.bath.ac.uk/~jjb/web/bod.html

Core idea:

1. Decompose behavior in a top-down fashion
2. Implement it bottom-up
3. Test, Revise, Reiterate

Procedural Scripting

UT2004 FPS Bot Example

```
public void logic() throws PogamutException {
    if (   weaponry.hasLoadedWeapon(UT2004ItemType.LIGHTNING_GUN)
        || weaponry.hasLoadedWeapon(UT2004ItemType.MINIGUN)) {
        if      (weaponry.hasLoadedWeapon(UT2004ItemType.LIGHTNING_GUN)) weaponry.changeWeapon(UT2004ItemType.LIGHTNING_GUN);
        else if (weaponry.hasLoadedWeapon(UT2004ItemType.MINIGUN))       weaponry.changeWeapon(UT2004ItemType.MINIGUN);
    }
    Player player = players.getNearestVisiblePlayer();
    if (player != null) {
        shoot.shoot(player);
        if (weaponry.hasLoadedWeapon(UT2004ItemType.LIGHTNING_GUN) || weaponry.hasLoadedWeapon(UT2004ItemType.MINIGUN)) {
            navigation.navigate(player);
            return;
        }
    }
    if (info.getHealth() < 50) {
        Item health = fwMap.getNearestItem(items.getSpawnedItems(UT2004ItemType.HEALTH_PACK).values(),
                                           info.getNearestNavPoint());
        if (health != null) {
            navigation.navigate(health);
            return;
        }
    }
    if (navigation.isNavigating()) return;
    Item weapon = null;
    for (ItemType weaponType : ItemType.Category.WEAPON.getTypes()) {
        if (weaponry.hasWeapon(weaponType)) continue;
        weapon = fwMap.getNearestItem(items.getAllItems(weaponType).values(), info.getNearestNavPoint());
        if (weapon != null) break;
    }
    if (weapon != null) {
        navigation.navigate(weapon.getNavPoint());
        return;
    }
    navigation.navigate(navPoints.getRandomNavPoint());
}
```

Procedural Scripting

UT2004 FPS Bot Example – BOD Applied

```
public void logicBOD() throws PogamutException {
    if (hasGoodLoadedWeapon()) {
        ensureGoodWeaponSelected();
    }
    if (hasAdversary()) {
        shootAdversary();
        if (hasGoodLoadedWeapon()) {
            pursueAdversary();
            return;
        }
    }
    if (hasLowHealth() && knowHealthLocation()) {
        runForHealth();
        return;
    }
    if (canGetNewWeapon()) {
        runForNewWeapon();
        return;
    }
    runRandomly();
}
```

Procedural Scripting

More Actions in Less Reactive Example

Procedural Scripting

The Effect of Durative Actions

```
private static enum ActionResult { RUNNING, FAIL, DONE };

private int actionIndex = 0;

public ActionResult logicSequence() throws PogamutException {
    switch(actionIndex) {
    case 0:
        switch (goToDog()) {
        case RUNNING: return ActionResult.RUNNING;
        case FAIL:    actionIndex = 0; return ActionResult.FAIL;
        case DONE:    actionIndex = 1; break;
        }
    case 1:
        switch (crouch()) {
        case RUNNING: return ActionResult.RUNNING;
        case FAIL:    actionIndex = 0; return ActionResult.FAIL;
        case DONE:    actionIndex = 2; break;
        }
    case 2:
        switch (petTheDog()) {
        case RUNNING: return ActionResult.RUNNING;
        case FAIL:    actionIndex = 0; return ActionResult.FAIL;
        case DONE:    actionIndex = 0; return ActionResult.DONE;
        }
    }
    return ActionResult.FAIL;
}
```

Procedural Scripting

OOP Style

```
private Sequence petTheDogSequence = new Sequence( goToDog, crouch, petTheDog );

public ActionResult logicSequence2() throws PogamutException {
    return petTheDogSequence.execute();
}
```

Procedural Scripting

OOP Style

```
private Action eatFoodFromFridge    = new Eat(foodFromFridge);
private Action eatFoodFromRestaurnt = new Eat(foodFromRestaurant);
private Sequence petTheDogSequence  = new Sequence(goToDog, crouch, petTheDog);
private Sequence eatingOutSequence  = new Sequence(goToRestaurant, orderFood, eatFoodFromRestaurnt);

private
    Root root = new Root(
                    new PrioritySwitch(
                        new SwitchItem( seeFriend, new Sequence(goToFriend, hailFriend, talkToFriend) ),
                        new SwitchItem( seeDog,    petTheDogSequence),
                        new SwitchItem( isHungry,
                            new Decision(
                                new DecisionItem(
                                        atHome,
                                            new Sequence(goToFridge, takeFoodFromFridge, eatFoodFromFridge)
                                ),
                                new DecisionItem( out, eatingOutSequence )
                            )
                        )
                    )
                );

public void logic3() throws PogamutException {
    root.execute();
}
```

Procedural Scripting

How do we perceive the code?

What does this do?

```
A := 1;
B := A + 1;
write(B);
```

No magic.

```
A := 1;
B := A + 1;
write(B);
```

Outputs “2”.

What does this do?

```
socket = acceptConnection();
socket.sendLine(“HELLO”);
line = socket.readLine();
```

Bit of magic

```
socket = acceptConnection();
socket.sendLine(“HELLO”);
line = socket.readLine();
```

acceptConnection() hangs the thread and waits for client connection, than it continues by greeting the client and again hangs the thread when waiting for reply.

What does this do?

```
goToDog();
crouch();
hugTheDog();
```

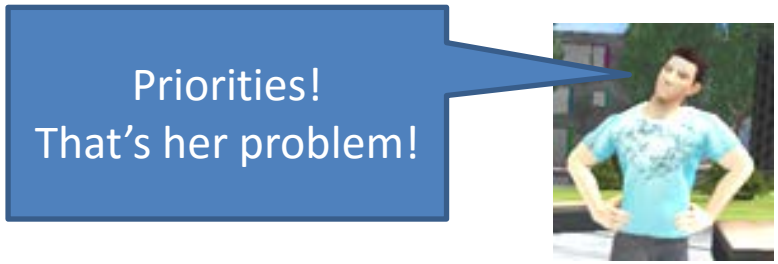
Well… sequence of actions?

```
goToDog();
crouch();
hugTheDog();
```

Well there is no reason to start petting the dog if we did not get to it first, this must be a sequence => respective methods then must act the same way as the acceptConnection(); hanging the thread.

What does this do?

```
If (inDanger()) getAway()
else
If (seeCuteDog()) petTheDog()
else
If (seeAPerson() && inTheMood()) hangOut()
else wanderAround()
```



Can we make these snippets of code reusable by putting them into procedures/methods?

Let’s see…

```
A := 1;
B := A + 1;
write(B);
```

No problem here.

```
procedure StrangeProc();
begin
    var A: integer;
    var B: integer;
    A := 1;
    B := A + 1;
    write(B);
end;
```

Let’s see

```
socket = acceptConnection();
socket.sendLine(“HELLO”);
line = socket.readLine();
```

This works too.

```
void waitForConnection() {
    Socket socket;
    String line;
    socket = acceptConnection();
    socket.sendLine("HELLO");
    line = socket.readLine();
}
```

This works too.

```
void waitForConnection() {
    Socket socket;
    String line;
    socket = acceptConnection();
    socket.sendLine("HELLO");
    line = socket.readLine();
}
```

We need thread-pooling right, but still okey (don't mind NIOs for a while).

How about this sequence of actions?

```
goToDog();
crouch();
hugTheDog();
```

Yeah, still working....

```
void petTheDog () {
    goToDog();
    crouch();
    hugTheDog();
}
```

Yeah, still working...

```
void petTheDog () {
    goToDog();
    crouch();
    hugTheDog();
}
```

We have to treat it the same way as the server thread from previous example.

What about this?

```
If (inDanger()) getAway()
else If (seeCuteDog()) petTheDog()
else If (seeAPerson() && inTheMood())
                hangOut()
else wanderAround()
```

Is this working?

```
void freeTime() {
  If (inDanger()) getAway()
  else If (seeCuteDog()) petTheDog()
  else If (seeAPerson() && inTheMood())
                hangOut()
  else wanderAround()
}
```

Let me see…

```
void freeTime() {
  If (inDanger()) getAway()
  else If (seeCuteDog()) petTheDog()
  else If (seeAPerson() && inTheMood())
                hangOut()
  else wanderAround()
}
```

During the first call, it happened, that she was not in danger but she saw a cute dog …
… so she stuck somewhere within petTheDog().

THIS DOES NOT WORK!

```
void freeTime() {
  If (inDanger()) getAway()
  else If (seeCuteDog()) petTheDog()
  else If (seeAPerson() && inTheMood())
                hangOut()
  else wanderAround()
}
```

Then the dog turned into a freakish monster!
… but she hugged it nevertheless.

Procedural Scripting

The Switching does not play well with Durative Actions

```
void freeTime() {
  If (inDanger()) getAway()
  else If (seeCuteDog()) petTheDog()
  else If (seeAPerson() && inTheMood())
          hangOut()
  else wanderAround()
}
```

1.  

petTheDog()

2. 

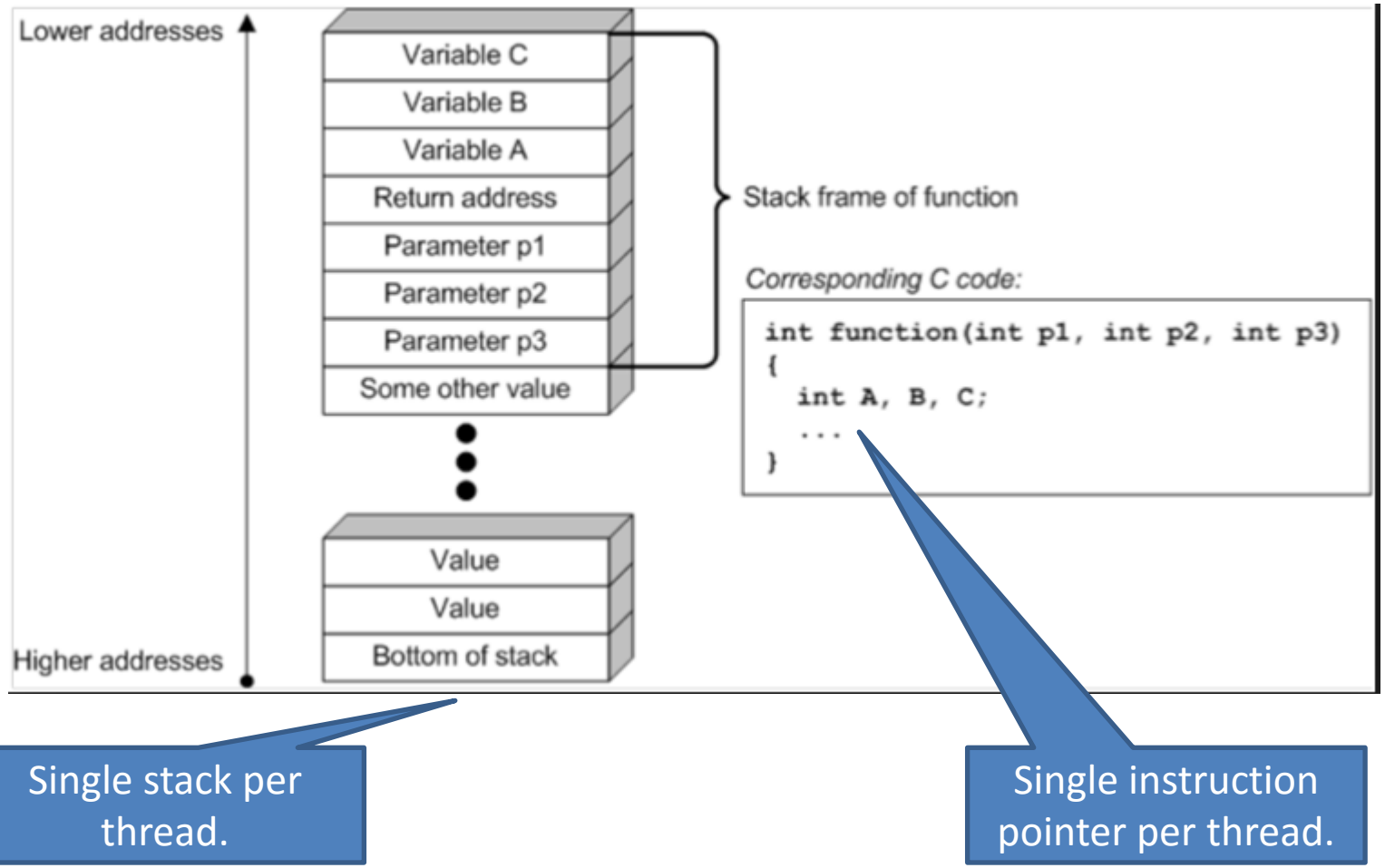
petTheDog()

3. 

getAway()

Procedural Scripting

Stack-based code representation



Reactive Planning

Procedural Scripting… No Good?

vs.

- Highly reactive
- Only a few (parameterized) easily interruptible actions

=> PS is OK

- Also Reactive
- Lot of actions
- Lot action sequences that must be managed

=> PS is not a good choice

Reactive Planning

Procedural Scripting… What else?

1. FSM-based techniques
   - "No" stack
   - Shifting locality of decision making process
2. Tree-based techniques
   - + "Stack-traversing"
3. BDI-like
   - Multiple-stacks, Blackboard-based

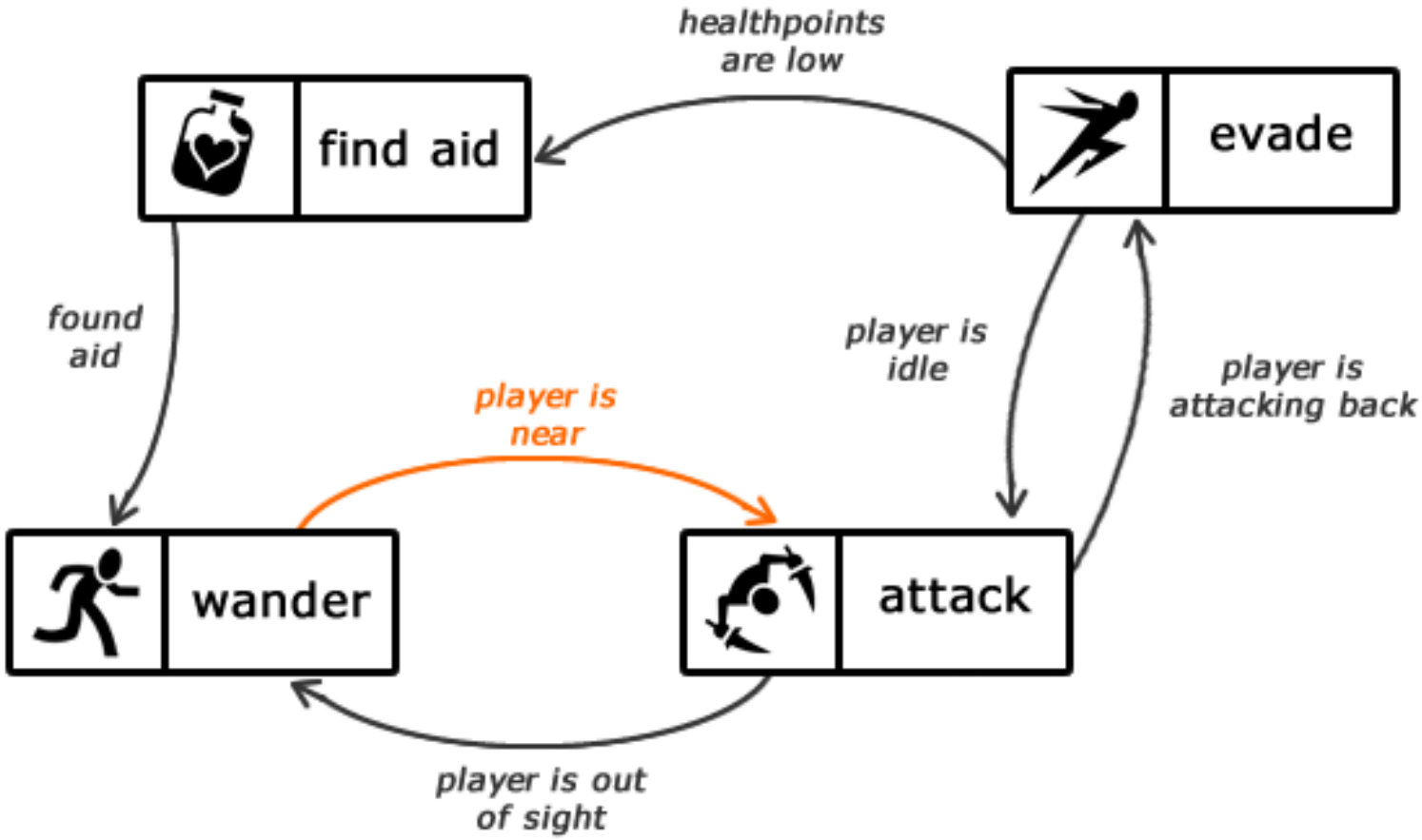
Reactive Planning

Procedural Scripting… What else?

1. **FSM-based techniques**
   - "No" stack
   - Shifting locality of decision making process
2. Tree-based techniques
   - + "Stack-traversing"
3. BDI-like
   - Multiple-stacks, Blackboard-based

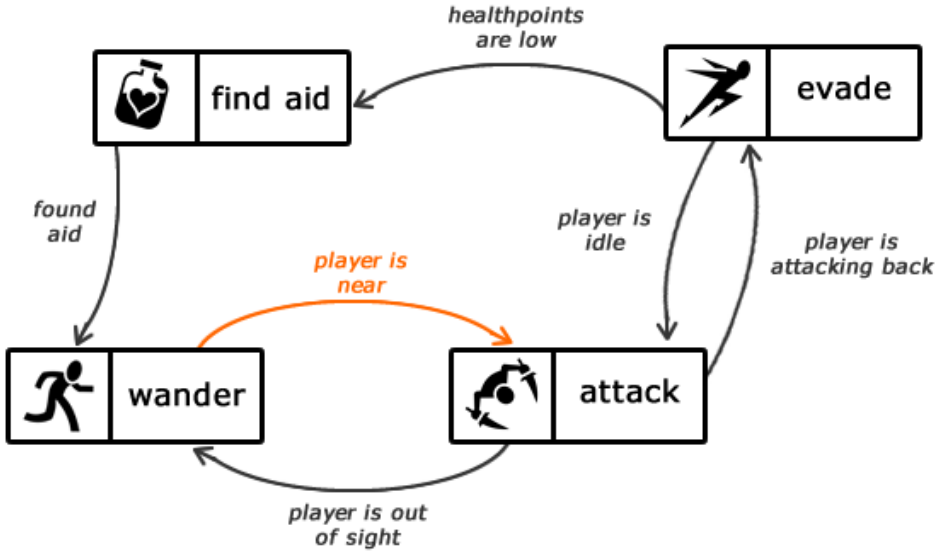
Reactive Planning

Finite State Machine



Finite State Machines
Example

- "No" stack … just "a single state"
- Decision making is made "within state"
  - That's why is this Turing complete as well!
- Very fast execution, easy to implement, can be visualized



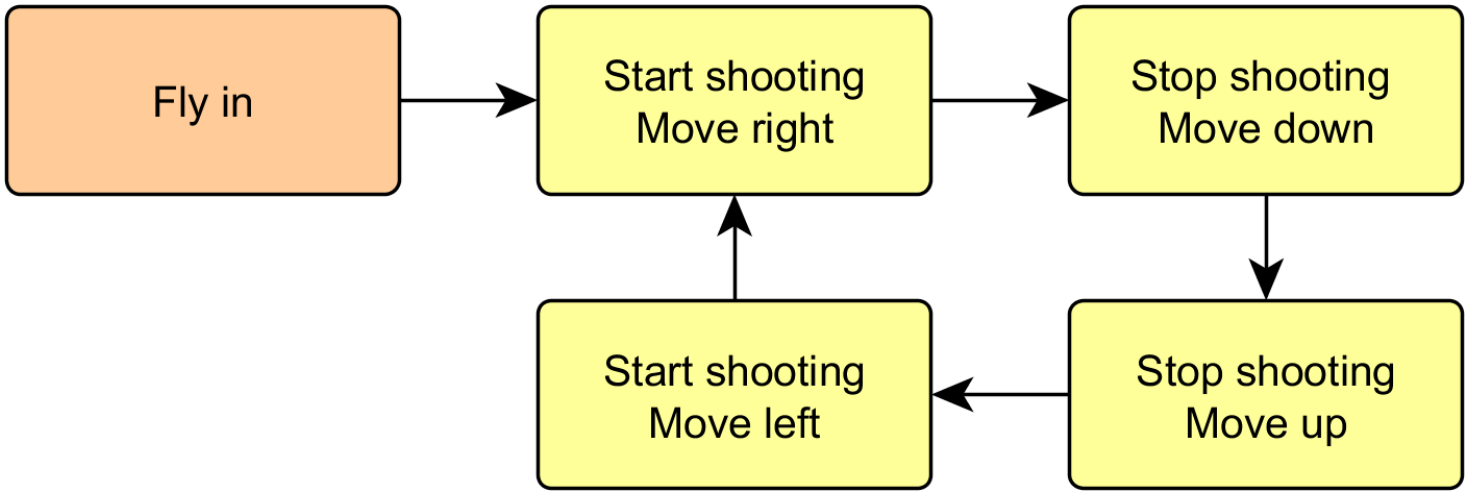
Finite State Machines

Classic tool of Game AI!

https://www.youtube.com/watch?v=HRDc3dSKFeA (4:40)

Finite State Machines

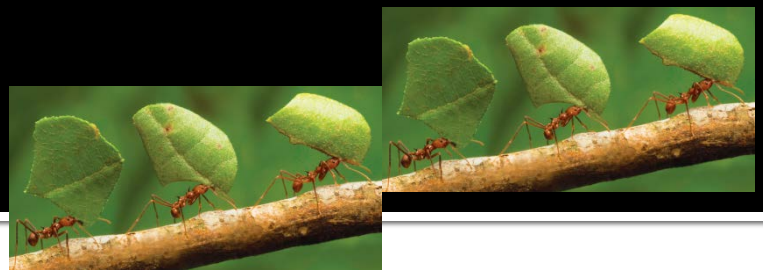
Classic tool of Game AI!



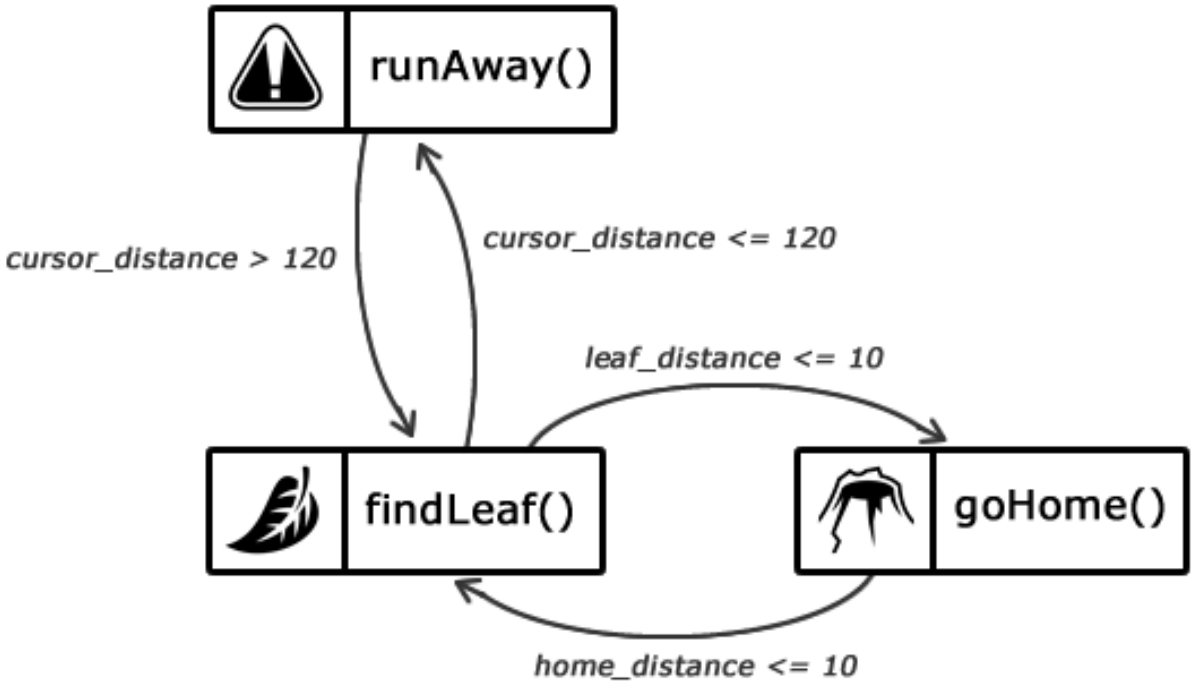
- What is missing?
  - "Almost dead" animation
- How to integrate it in there?
  - Ad-hoc code that lies "somewhere"

Finite State Machines

Another example

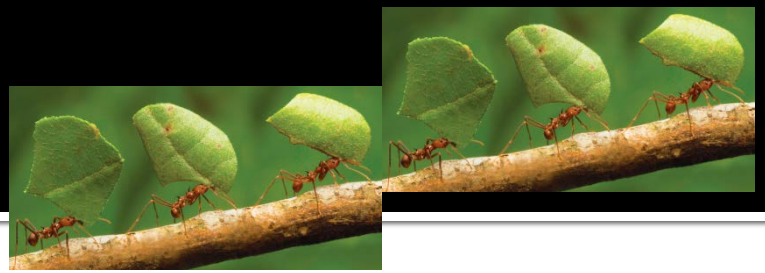


Ants are foraging in 2D cage trying to find leaves while avoiding your cursor.

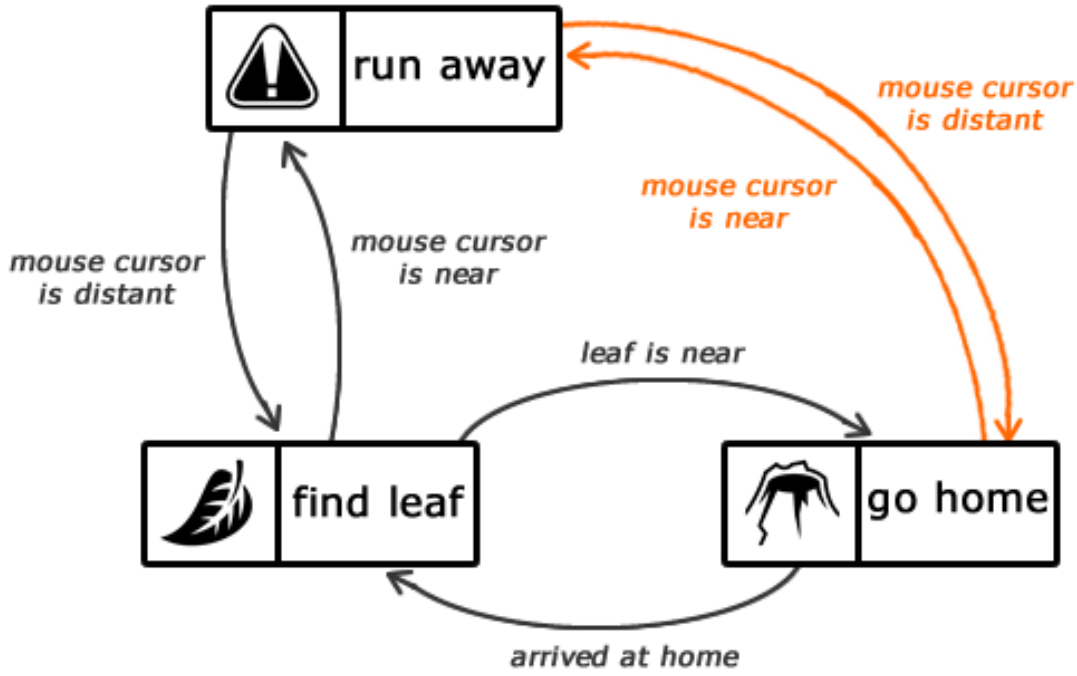


Finite State Machines

Another example

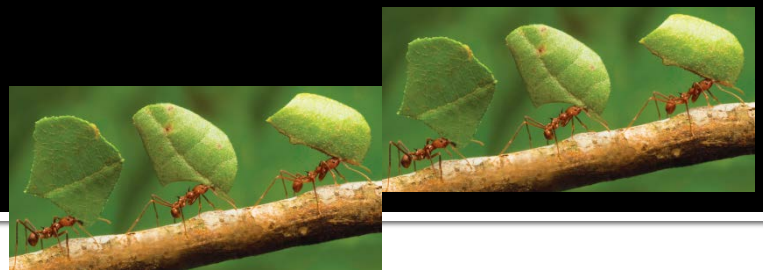


However, we have "switching-problem" here!
Priorities!



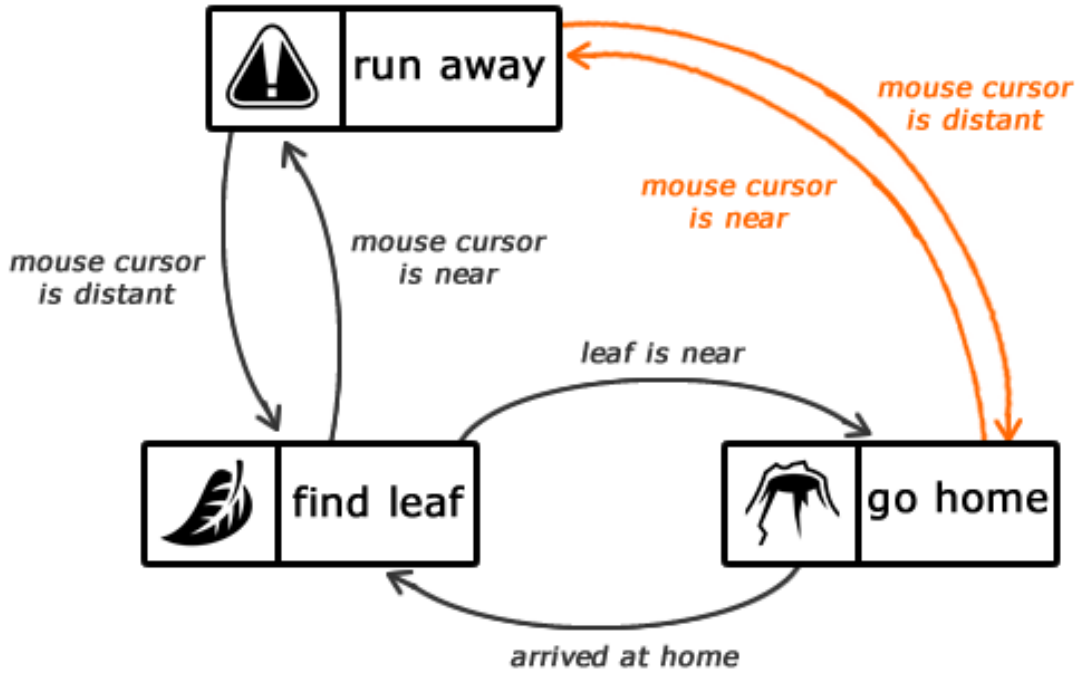
Finite State Machines

Another example

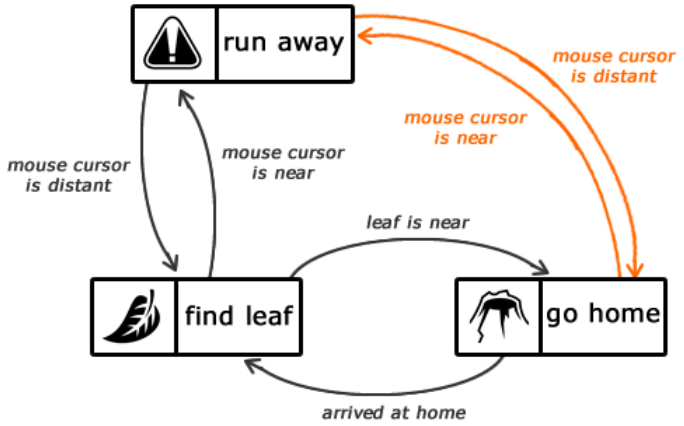


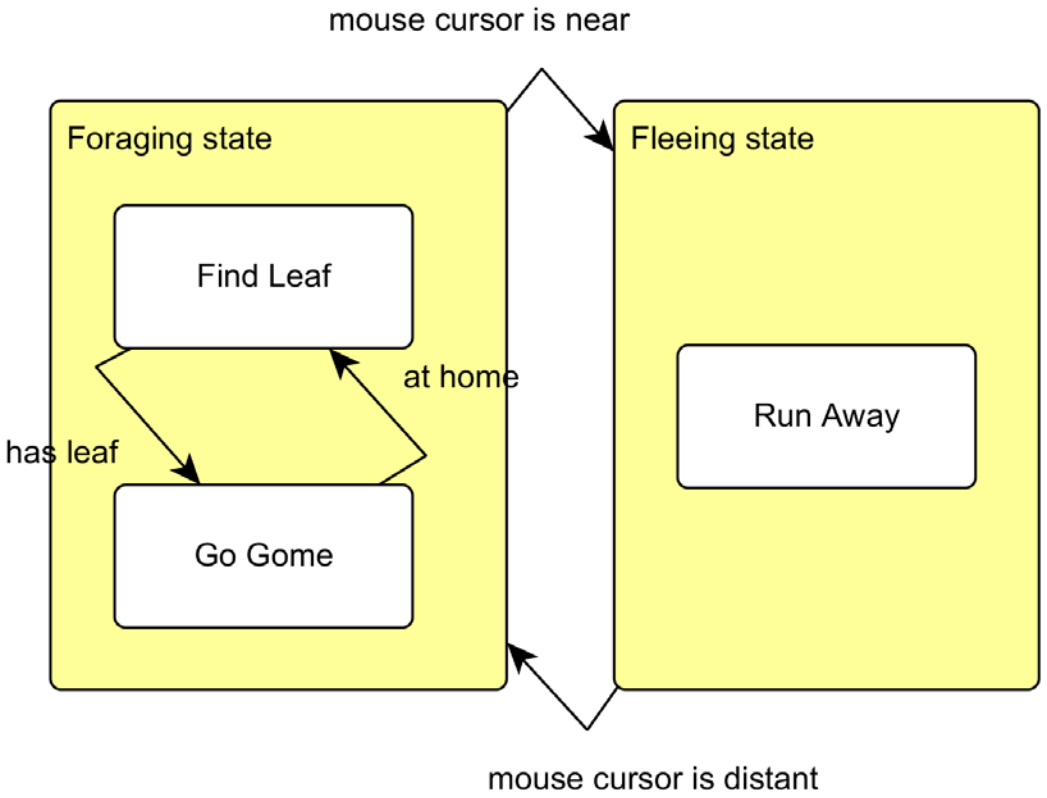
Any "FSM-like" solution to this?

=> Hierarchical FSMs



Finite State Machines
Hierachical



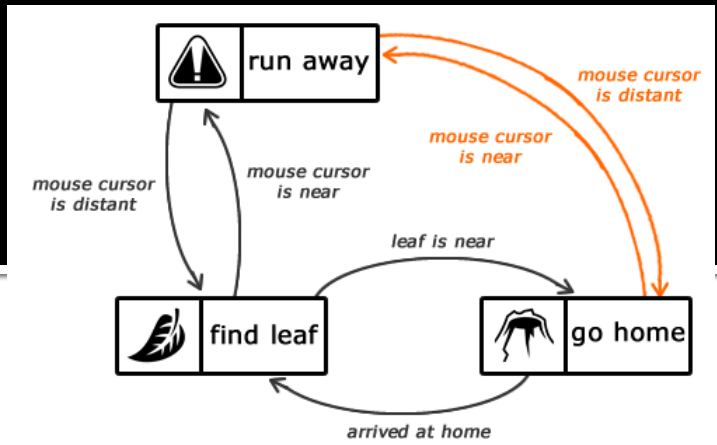


Looking good?

Why we do not just…

Finite State Machines

Stack-based

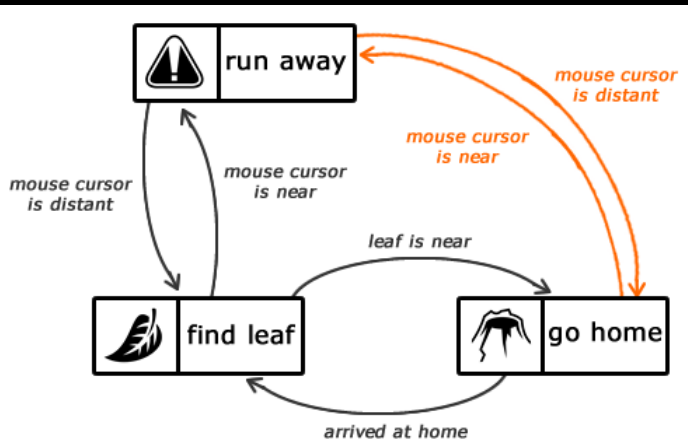


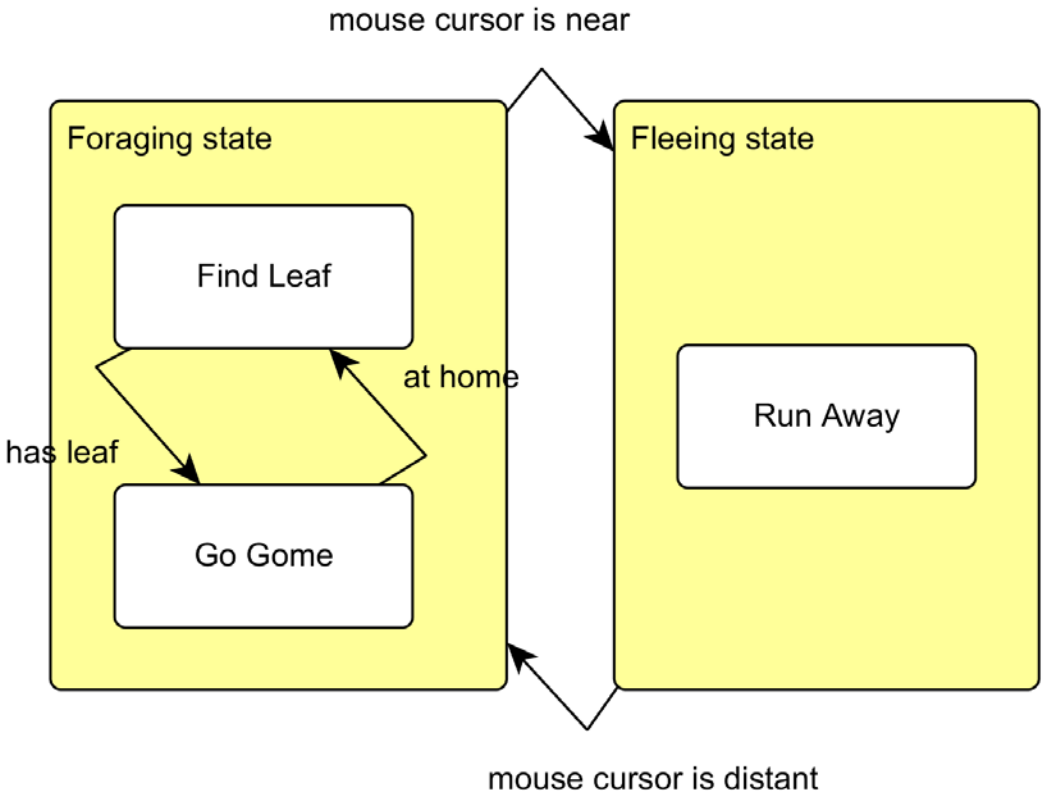

```
public void antLogic() {

    if (mouseCursorNear()) {
        runAway();
        return;
    }
    if (hasLeaf()) {
        goHome();
        return;
    }
    if (atHome()) {
        dropTheLeaf();
        return;
    }
}
```

Gotcha!

Finite State Machines
Hierachical





Still okayish…

But What about our FPS bot…

Finite State Machines
Hierachical

Can we model this with hFSM?

```
public void logicBOD() throws PogamutException {
    if (hasGoodLoadedWeapon()) {
        ensureGoodWeaponSelected();
    }
    if (hasAdversary()) {
        shootAdversary();
        if (hasGoodLoadedWeapon()) {
            pursueAdversary();
            return;
        }
    }
    if (hasLowHealth() && knowHealthLocation()) {
        runForHealth();
        return;
    }
    if (canGetNewWeapon()) {
        runForNewWeapon();
        return;
    }
    runRandomly();
}
```

Parallel action

Parallel action

Priority list with 4 elements

Finite State Machines

Sequential Hierachical

Def. 13. Sequential Finite-state Machine (sFSM) Behavior

Definition: (States, init, script)

States	A non-empty finite set of sFSM states.
init	∈ **S,** an initial state.
script	Associated sFSM script used by sFSM activities.

Runtime: (state)

state	Current state of sFSM, initial value: **init**.

No "accepting" state(s) … not needed

Finite State Machines

Sequential Hierachical

Def. 14. Sequential Finite-state Machine State

(T, A)

T An ordered list of transitions.

A transition is a triple **(c, s, a):**

- **c** a transition condition (boolean expression),
- **s** ∈ **States** is target sFSM,
- **a** is an **OnTransition** activity.

A A triple of activities (**OnEnter**, **OnInternal**, **OnExit**).

May use reasoning!

Activities are Turing-complete

Finite State Machines

Sequential Hierachical

Alg. 2. Sequential Finite-state Machine Action-Selection

```
      procedure sFSM-ASM (sFSM)
01:      for each (Transition t in sFSM.state.T)
02:         if (t.c)
03:            sFSM.state.OnExit.run()
04:            t.OnTransition.run()
05:            t.s.OnEnter.run()
06:            sFSM.state = t.s
07:            return
08:      sFSM.state.OnInternal.run()
```

Here we can reset nested sFSM state

Hierarchical variant via nesting sFSM

Finite State Machines
Sequential Hierachical

Can we model this with shFSM?

```
public void logicBOD() throws PogamutException {
    if (hasGoodLoadedWeapon()) {
        ensureGoodWeaponSelected();
    }
    if (hasAdversary()) {
        shootAdversary();
        if (hasGoodLoadedWeapon()) {
            pursueAdversary();
            return;
        }
    }
    if (hasLowHealth() && knowHealthLocation()) {
        runForHealth();
        return;
    }
    if (canGetNewWeapon()) {
        runForNewWeapon();
        return;
    }
    runRandomly();
}
```

Parallel action

Parallel action

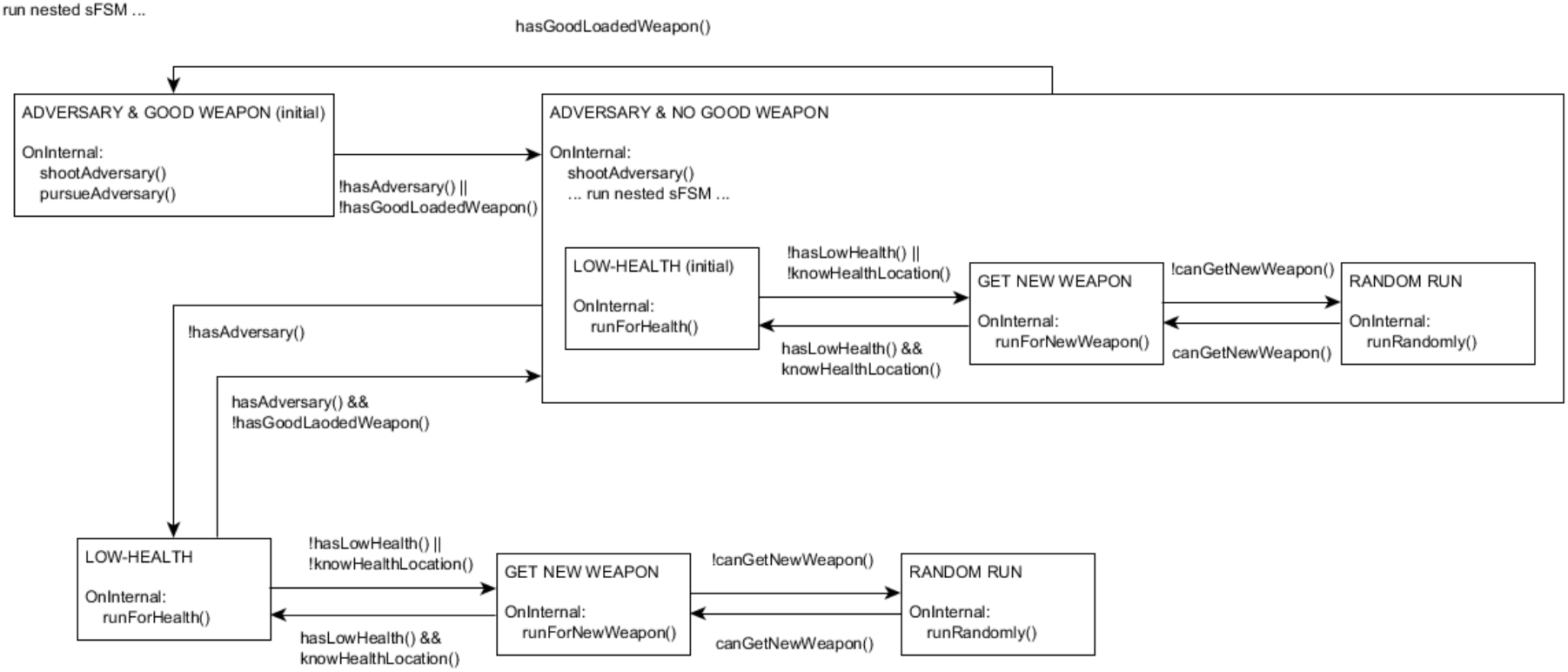
Priority list with 4 elements

Finite State Machines

Sequential Hierachical

logicBOD

OnInternal:
if (hasGoodLoadedWeapon())
ensureGoodWeaponSelected()
... run nested sFSM ...



Finite State Machines

Sequential Hierachical

logicBOD

OnInternal:
if (hasGoodLoadedWeapon())
ensureGoodWeaponSelected()
... run nested sFSM ...

hasGoodLoadedWeapon()

ADVERSARY & GOOD WEAPON (initial)

OnInternal:
shootAdversary()
pursueAdversary()

!hasAdversary() ||
!hasGoodLoadedWeapon()

ADVERSARY & NO GOOD WEAPON

OnInternal:
shootAdversary()
... run nested sFSM ...

hasLowHealth() &&
knowHealthLocation()

hasAdversary() &&
hasGoodLoadedWeapon()

LOW-HEALTH (initial)

OnInternal:
runForHealth()

!hasLowHealth() ||
!knowHealthLocation()

GET NEW WEAPON

OnInternal:
runForNewWeapon()

!canGetNewWeapon()

RANDOM RUN

OnInternal:
runRandomly()

canGetNewWeapon()

hasLowHealth() &&
knowHealthLocation()

!hasAdversary()

hasAdversary() &&
!hasGoodLaodedWeapon()

hasAdversary() &&
!hasGoodLaodedWeapon()

hasAdversary() &&
!hasGoodLoadedWeapon()

LOW-HEALTH

OnInternal:
runForHealth()

!hasLowHealth() ||
!knowHealthLocation()

GET NEW WEAPON

OnInternal:
runForNewWeapon()

!canGetNewWeapon()

RANDOM RUN

OnInternal:
runRandomly()

canGetNewWeapon()

hasLowHealth() &&
knowHealthLocation()

hasLowHealth() &&
knowHealthLocation()

hasAdversary() &&
hasGoodLoadedWeapon()

hasAdversary() &&
hasGoodLoadedWeapon()

Finite State Machines
Sequential Hierachical

Can we model this with shFSM?

```
public void logicBOD() throws PogamutException {
    if (hasGoodLoadedWeapon()) {
        ensureGoodWeaponSelected();
    }
    if (hasAdversary()) {
        shootAdversary();
        if (hasGoodLoadedWeapon()) {
            pursueAdversary();
            return;
        }
    }
    if (hasLowHealth() && knowHealthLocation()) {
        runForHealth();
        return;
    }
    if (canGetNewWeapon()) {
        runForNewWeapon();
        return;
    }
    runRandomly();
}
```

Parallel action

Parallel action

Priority list with 4 elements

Yes!

But certainly we DO NOT WANT TO!

Finite State Machines
Other modifications

- Fuzzy transitions
  - Using fuzzy variables for conditions
- Probabilistic FSM
  - Choosing random transition

⇒ Shares the same characteristics

- ⇒ Great for "sequences with choice points"
- ⇒ Bad for reactive stuff that has to arbitrate between multiple (>3) priorities or having parallel actions

Reactive Planning

Procedural Scripting… What else?

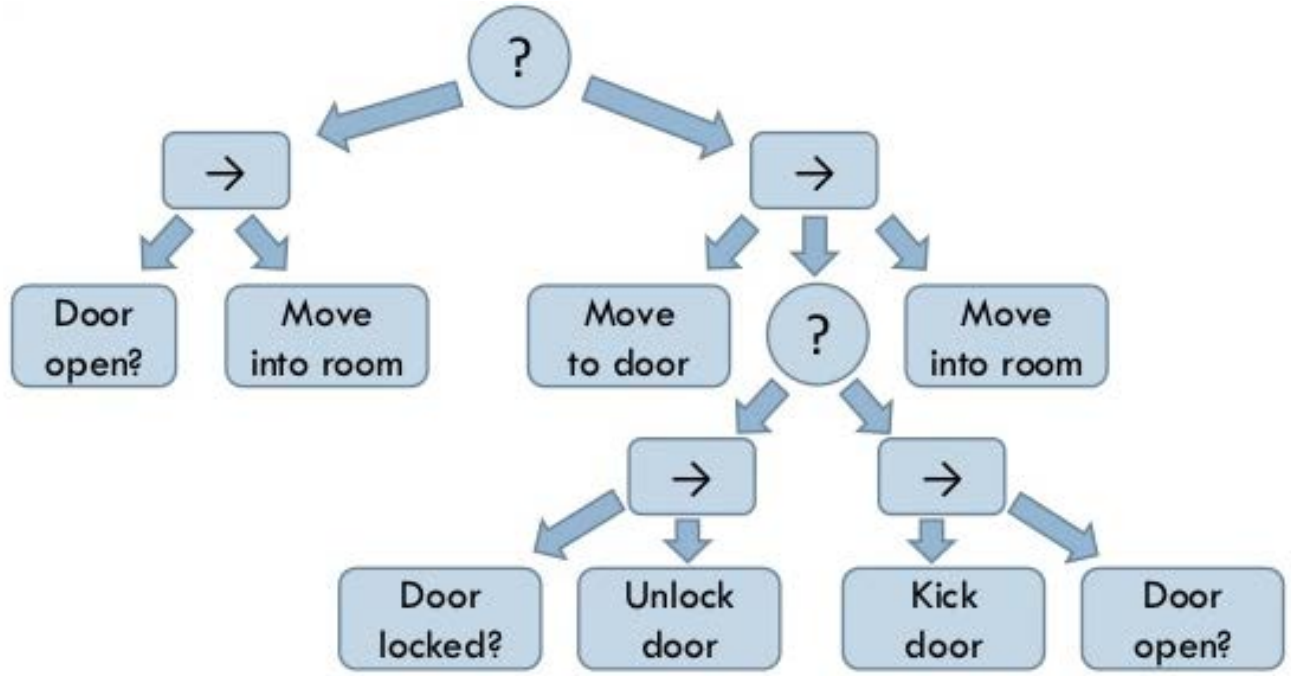
1. FSM-based techniques
   - "No" stack
   - Shifting locality of decision making process
2. **Tree-based techniques**
   - + "Stack-traversing"
3. BDI-like
   - Multiple-stacks, Blackboard-based

Reactive Planning

Behavior Trees

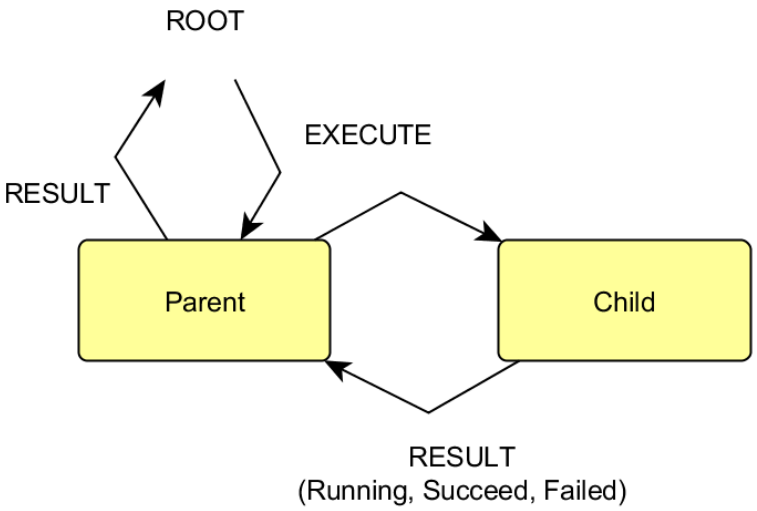
http://www.slideshare.net/StavrosVassos/aigames-lecture1part2

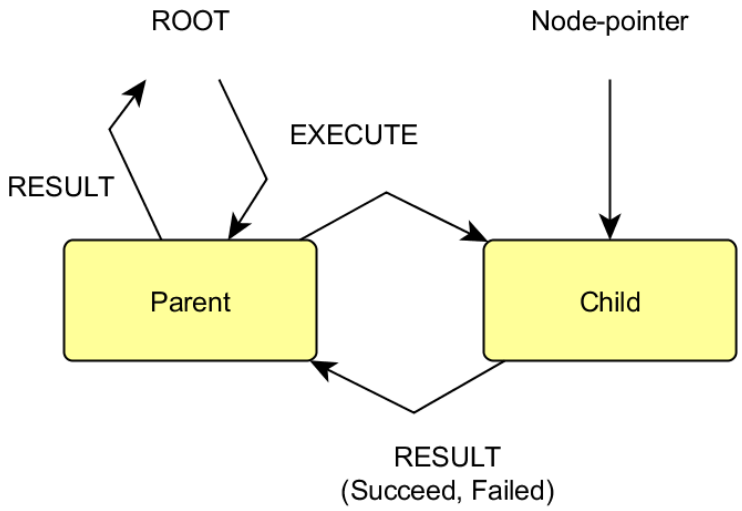
Slides 14-18



Reactive Planning
Behavior Trees

- Two types of BTs
  - With "node-pointer"
  - Root traversal





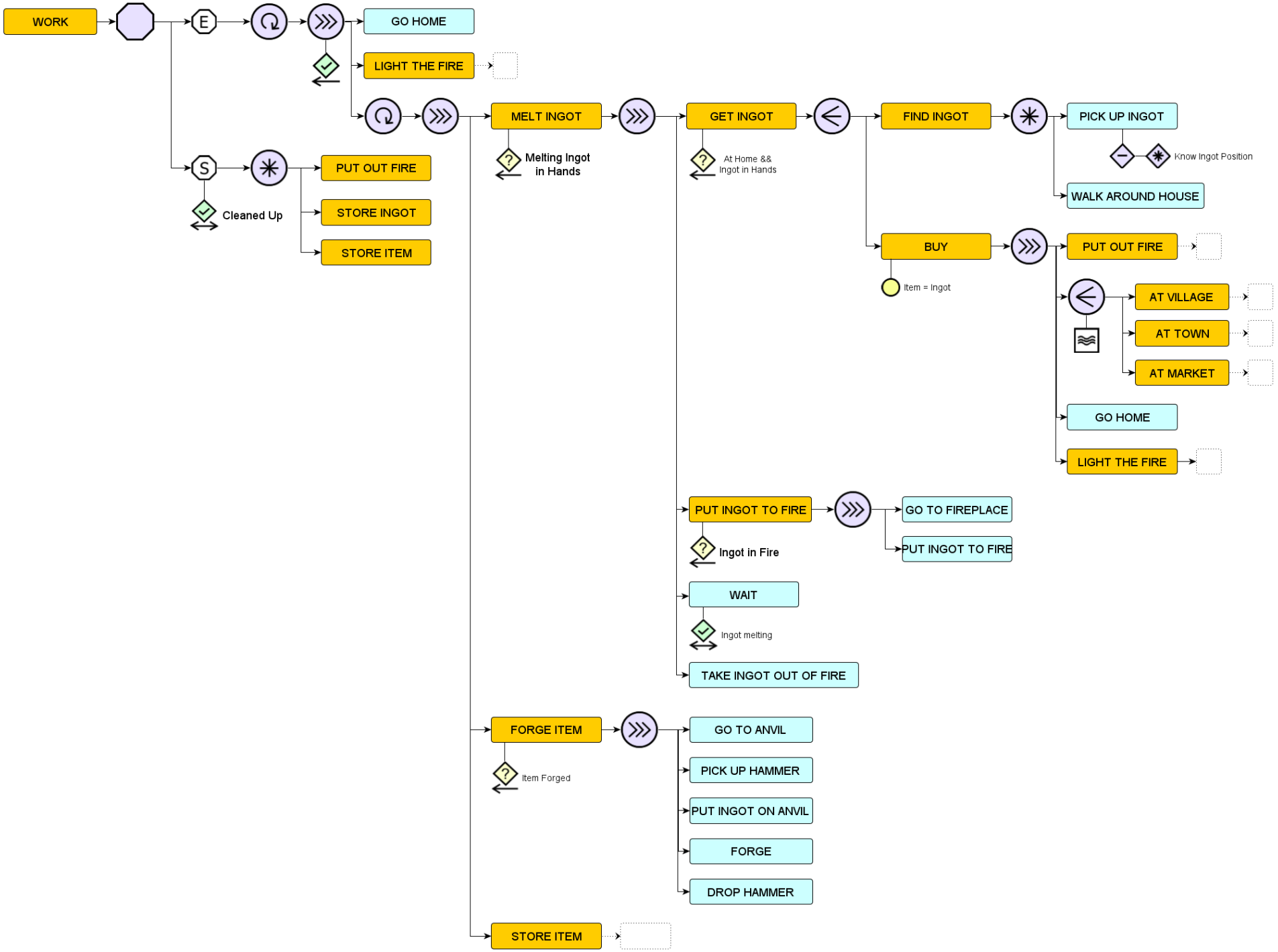
Reactive Planning
Behavior Trees

- Lot of node-types
  - Sequence, Selector, Loop, Parallel
  - Gates (condition, count-based, time-based, dis/enabler)
  - Reinterpret the result
- Possible behavior coordination
  - N agents "executes" the same tree
  - Gates limiting number of agents
  - Join-nodes (agent is waiting for coordination)

Reactive Planning

Behavior Trees

- Root traversal trees ~ “Stack-traversing”
- Blacksmith example


WORK
E
GO HOME
LIGHT THE FIRE
S
PUT OUT FIRE
Cleaned Up
STORE INGOT
STORE ITEM
MELT INGOT
Melting Ingot in Hands
GET INGOT
At Home && Ingot in Hands
FIND INGOT
PICK UP INGOT
Know Ingot Position
WALK AROUND HOUSE
BUY
Item = Ingot
PUT OUT FIRE
AT VILLAGE
AT TOWN
AT MARKET
GO HOME
LIGHT THE FIRE
PUT INGOT TO FIRE
Ingot in Fire
GO TO FIREPLACE
PUT INGOT TO FIRE
WAIT
Ingot melting
TAKE INGOT OUT OF FIRE
FORGE ITEM
Item Forged
GO TO ANVIL
PICK UP HAMMER
PUT INGOT ON ANVIL
FORGE
DROP HAMMER
STORE ITEM

Reactive Planning
Behavior Trees

- Root traversal trees ~ "Stack-traversing"
- That's what standard **If-Then rules** are doing!
  - But the stack lacks "statefulness"

⇒ Are Behavior Trees to Procedural Scripting what SQL is to Cobol?

Reactive Planning

Procedural Scripting… What else?

1. FSM-based techniques
   - "No" stack
   - Shifting locality of decision making process
2. Tree-based techniques
   - + "Stack-traversing"
3. **BDI-like**
   - Multiple-stacks, Blackboard-based