Behavior Trees and Reactive Planning

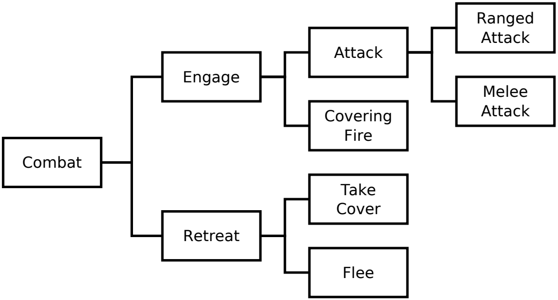
Peter Mawhorter

October 8, 2010

What is a Behavior Tree?

- A tree:
  - Leaf nodes are primitive behaviors ("throw grenade").
  - Internal nodes are abstract behaviors ("attack").
- This tree is evaluated to decide on a behavior.

An Example


Combat
Engage
Retreat
Attack
Covering Fire
Take Cover
Flee
Ranged Attack
Melee Attack

What About FSMs?

- In a finite state machine, state transitions are explicit
- In a behavior tree, state preconditions are explicit
- A simple FSM corresponds to a behavior list, a behavior tree is the equivalent of an HFSM
- Preconditions are less numerous than transitions

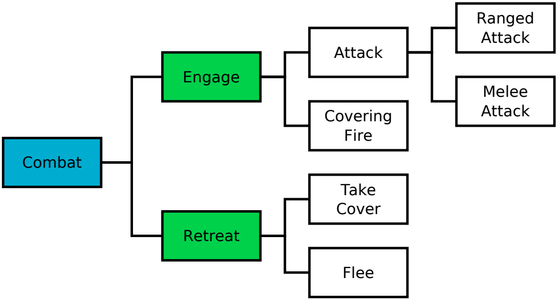
Behavior Selection

- Each behavior has preconditions which determine which can be selected
- Starting with the root, each behavior picks one of its available children to run
- This choice can be random or prioritized, or may use some other scheme
- Once a leaf is chosen, that concrete behavior is activated
- When a behavior finishes, behavior selection starts over again at the root

Success and Failure

- Primitive behaviors may succeed or fail
- Higher-level behaviors depend on their children to succeed
- Failure may cause the parent to select an alternate child instead of failing immediately

A Behavior Tree in Action



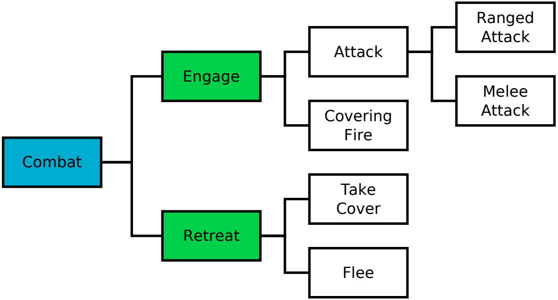
A Behavior Tree in Action

```
behavior Engage {
  preconditions (
    health > 10%
  and
    have_weapon
  )
  children (
    Attack
    Covering_Fire
  )
}
```

```
behavior Retreat {
  preconditions (
    health < 50%
  or
    outnumbered
  )
  children (
    Take_Cover
    Flee
  )
}
```

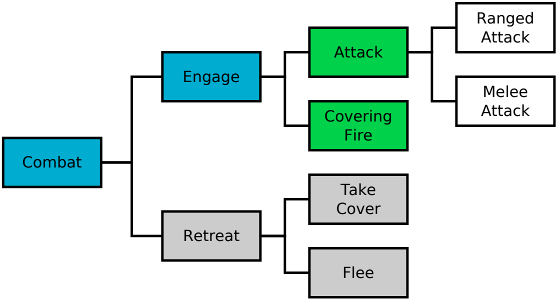
A Behavior Tree in Action

- health: 90%, weapon: `rifle`, outnumbered: `false`



A Behavior Tree in Action

- health: 90%, weapon: `rifle`, outnumbered: `false`



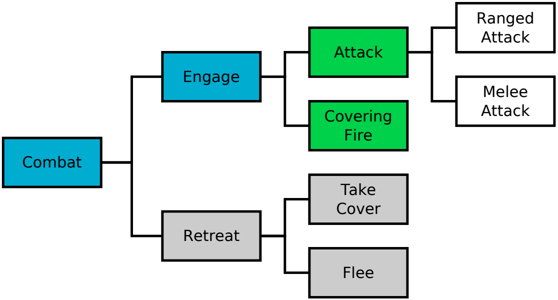
A Behavior Tree in Action

```
behavior Attack {
  preconditions (
    have_weapon
  )
  children (
    Ranged_Attack
    Melee_Attack
    Throw_Grenade
  )
}
```

```
behavior Covering_Fire {
  preconditions (
    have_ranged_weapon
  and
    ally_under_fire
  )
  action (
    Covering_Fire_Action
  )
}
```

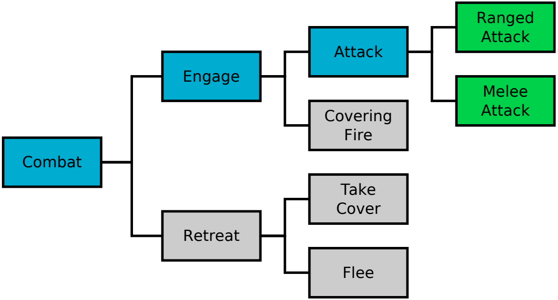
A Behavior Tree in Action

- weapon: `rifle`, ally_under_fire: `false`



A Behavior Tree in Action

- weapon: `rifle`, ally_under_fire: `false`



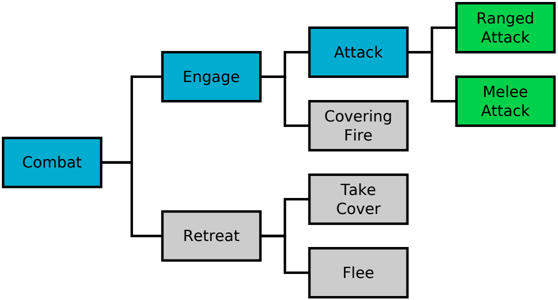
A Behavior Tree in Action

```
behavior Ranged_Attack {
  preconditions (
    have_ranged_weapon
  and
    opponent_in_range
  )
  action (
    Ranged_Attack_Action
  )
}
```

```
behavior Melee_Attack {
  preconditions (
    have_melee_weapon
  and
    opponent_in_melee
  )
  action (
    Melee_Attack_Action
  )
}
```

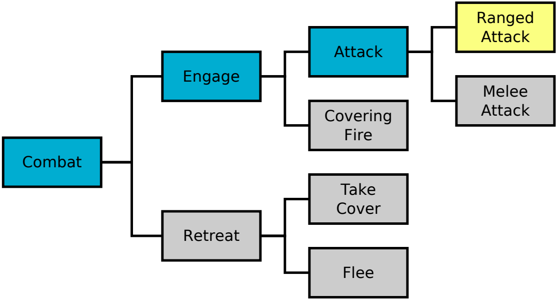
A Behavior Tree in Action

- weapon: `rifle`, opponent_range: `20m`



A Behavior Tree in Action

- weapon: `rifle`, opponent_range: `20m`



Some Caveats

- Rather than selecting a single child behavior, a node might run its children sequentially or in parallel
- Besides preconditions, a behavior might have context conditions
- High-priority behaviors might preempt low-priority ones
- In some systems, multiple behaviors might run at the same time
- Behavior selection can happen in response to an event

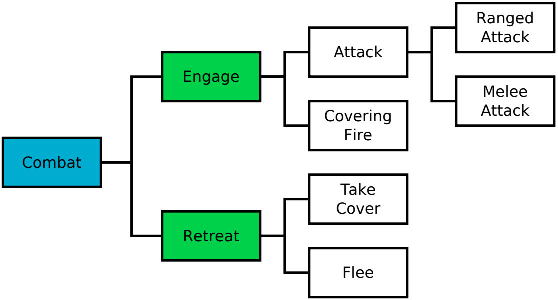
Behavior Priorities

```
behavior Engage {
  preconditions (
    health > 10%
  and
    have_weapon
  )
  priority (
    10
  )
  children (
    Attack
    Covering_Fire
  )
}
```

```
behavior Retreat {
  preconditions (
    health < 50%
  or
    outnumbered
  )
  priority (
    5 + (.5 - health%)*20 +
      (outnumbered ? 10 : 0)
  )
  children (
    Take_Cover
    Flee
  )
}
```

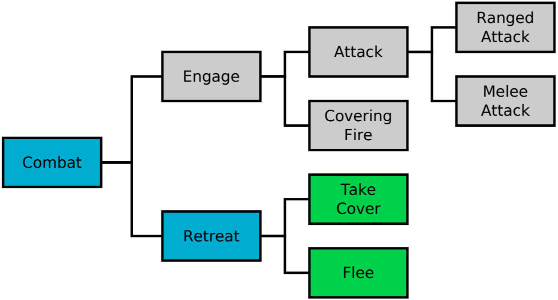
Behavior Priorities

- health: 20%, weapon: `rifle`, outnumbered: `false`
- Priorities: Engage: 10, Retreat: 11



Behavior Priorities

- health: 20%, weapon: `rifle`, outnumbered: `false`
- Priorities: Engage: 10, Retreat: 11



Behavior Trees in Halo 2

- Prioritized child selection drives most choices
- Impulse behaviors add some dynamic links to the tree
- Precondition checks are optimized using behavior tags
- Event-driven impulses allow more dynamic behavior
- Behavior options are limited via styles

Query-Enabled Behavior Trees

- Dynamic selection of child behaviors
- Uses case-based reasoning to select behavior candidates at runtime
- Selection is based largely on the variables used within the behaviors
- This is equivalent to making all of the cases children of each query node and performing prioritized selection at the query node using the case similarity metric

Reactive Planning

- Corresponds to a behavior tree that uses asynchronous selection, with all sorts of details thrown in
- While traditional planning uses an algorithm to search the space of all possible plans, reactive planning relies on the architect to describe the space of all permitted plans
- A reactive planner then selects eagerly and randomly from actions within this plan space, and tries something different whenever anything fails

Reactive Planning Example

```
sequential behavior vultureAttack(PlayerUnitWME vulture)
{
  int vultureID, ex, ey;

  with (success_test {
    (vulture.getHasTask()==false &&
     vulture.getOrder()==PlayerGuard)
    query = (UnitQueryWME fresh==true)
    (query.setIsEnemy(true))
    (query.setLocationUnit(vulture.getID()))
    (query.setIsGround(true))
    (UnitQueryWME nearest::enemyID)
    (EnemyUnitWME ID==enemyID realX::ex realY::ey)
  }) wait;
...
```

Reactive Planning Example (continued)

```
...

  mental_act {
    vulture.hasTask();
    vultureID = vulture.getID();
  }

  // attack and wait for the command to be issued
  act attackMovePixel(vultureID, ex, ey);
  subgoal WaitFrames(1);
}
```

Advantages of Behavior Trees

- Practical and intuitive
- More scalable than finite state machines
- Afford fine-grained and dynamic control over behavior

Disadvantages of Behavior Trees

- Coordination of multiple agents can be difficult
- Control is implicit, so bugs can be hard to understand and to fix
- Require some optimizations to fit into modern games
  - This is why full reactive planning for game agents would be difficult

Discussion Topics

- Questions?
- Are there 'tradeoffs' between behavior trees and reactive planning? What would you need to consider if you were building a game and deciding between them?
- Compared to FSMs, what do BTs make easy? What do they make hard?
- What dictates the structure of a behavior tree? In terms of design, what are the driving concerns?
- Are there other ways to solve the problems brought up by the query-enabled BTs paper?