FLINCH – An AI Entity Learning Algorithm

Walter B. Gress V

Drexel University

January 14, 2016

The following document details a concept for an algorithm called FLINCH. FLINCH is designed to teach individual soldiers over time in an RTS environment to perform the optimal operation given a situation.

# introduction

As said in the abstract, FLINCH is designed to teach individual units in an RTS to make the optimal decision given a situation. It does by using a behavior tree, fuzzy logic, state machines, and back propagation on the behavior tree along with storage in a running relational database.

# LIST OF FUNCTIONAL COMPONENTS

FLINCH is composed of:

* A decision/behavior tree
* Fuzzy Logic Deciders
* A\* Search to choose behaviors
* A Choice Database
* The running of a simulation to train decider’s fuzzy logic using backwards propagation on the behavior tree at least 100 times. The tree starts out completely balanced. As the simulations are run, scores are generated that effect which choices are made at the fuzzy logic/decider/finite state machine level.

# DECISION TREE

The calculation of a behavior starts at

the root if the tree and progresses through the tree according attributes until it reaches a behavior (leaf). (See Figure 1)

The State Machines are Deciders, where dynamic and or static stats are compared against the decision variables. So this is a tree structure. Different behaviors occur depending on the different statistics (sliders). Each decider has a different cost according to the comparison between the state machines variable and its own stats. The path with the lowest cost is the one that is chosen.

Here is an example:

Cost: Move Under Fire 0

Cost: State Machine 1 – Fuzzy Logic:::

The following

Player Loyalty Statistic 10%

Player Health Statistic: 20%

Cost: State Machine 2 – Fuzzy Logic::::

The following

Player Intelligence 90%

Player Engineering Training 50%

# A\* ALGORITHM

Use A\* algorithm over a decision tree with the beginning of the behavior as the root of the tree and its leaves leading to one or more behaviors. At each branch of the decision tree the decision is made in a state machine fashion, i.e, if the player is low on health but high on morale he is more likely to do the “Move under direct fire” than a player that is low on health and low on morale. I’m thinking about supervised learning and subsumption architcture but we may need to wait until the future to do that. Also I have to integrate the charts/stats from the original design document.

BEHAVIOR HAS MULTIPLE MOTIVES ( EXAMPLE: YOU WENT T O EAT BECAUSE IT IS 12:00 NOON, YOU ARE ON BREAK FROM CLASS, YOU DIDN’T EAT BREAKFAST.ETC

# DECISION TREE OUTLINE IN DETAIL

The decision tree is a simple rooted tree that starts out as a hard-coded number of neutral nodes and children. A behavior is attached to each leaf. When the AI is initialized, it creates this tree for a soldier, completely neutral. The soldier picks a random path and randomly moves down the tree until he or she reaches a leaf node where a behavior is. That behavior is executed and the results are either rewarded or punished depending on the outcome. The results are propagated backwards up the tree back to the root either strenghtening or weakining that behavior path.

# A\* Search and tree

Whenever an action is to be performed, the process begins by moving from the root down to the branch with the least g(x) path cost. The path cost is a combination of a trainable back-propagation variable, the same used to initially train the tree. The total decision is made of a fuzzy logic component and a series of comparisons of stats. The stats are compared to a set of master stats. Depending on the condition, the soldier’s g(x) score will go up or down. There are fifty conditions that are used in comparison with the fuzzy logic to determine if that branch will be traversed to the next node.

# BEHAVIOR FUNCTIONALITY

When a behavior (leaf) has been reached via the behavior tree, for example, our demo behavior *fireAtEnemies* which is of the behavior class, for all behaviors the Behavior Class function Behavior.run() is called.

Each Behavior class consists of a List of SubBehavior classes. Behavior.run() iterates through these SubBehavior classes.

The sub behavior classes are common amongst the Behavior classes. The sub behavior classes are hard-coded actions

that fulfill the Behavior. When behavior.run() iterates through each of its SubBehaviors in the list, for each SubBehavior there

is a simulate() function. The simulate function for the SubBehavior is a hard-coded action like, “Fire on Enemy”, “Dig in”, “Camoflauge”, Etc. The SubBehaviors are added to the list on Game Start.

For example:

(initialization)

…fireAtEnemies’ Behavior’s list is loaded with the SubBehaviors: Dig In, Camofluage, Open Fire

(In Game)

….A\* search reaches the leaf (behavior) fireAtEnemies

fireAtEnemies.run() is called

fireAtEnemies.run()

{

//list of subBehaviors in class

//List<SubBehaviors> subBehaviorsList

for (SubBehaviors sb in subBehaviorsList)

{

sb.simulate();

}

}

In this case, sb.simulate calls DigIn(), Camoflauge(), and OpenFire()

These sb.simulate calls, e.g., DigIn(), are hard coded.

# 

# BEHAVIORS

REWARD/PUNISH (BEHAVIOR AND RESULT)

The following behaviors have a behavior that belongs to one of three categories. The first, *Strategy*, is a handles such behaviors as navigating technology trees, build orders, upgrades, and army composition. In this whole bases and armies are considered. The second, *Tactics*, implements *Strategy*, encompasses confrontation, such as such as placement and movement of units, placement of buildings, exploiting terrain for troop placement,. The final, *Reactive Control*, how the player would act in real time. targeting enemy soldiers, firing, fleeing, hit and run techniques, intelligence gathering and general army movement.

Tactics and Reactive Control are handled by soldier’s AI as outlined below. Strategy elements are implemented via a separate *planner.*

Below is a list of high level behaviors, they were taken from various sources, including military manuals and websites related to the DoD. Following this list is a table that lists potential behaviors on the left and subbehaviors on the right. The subbehaviors are nitty gritty, they are a look on how the higher level behavior would be implemented.

|  |  |
| --- | --- |
| **Rewarded Behavior** | **SubBehaviors** |
| Direct Hit with Rifle | 1. Players Weapon is a Rifle 2. Player Maps Line of Site 3. Player Fires Weapon 4. Bullet Object is created 5. If Bullet Collides calculate   damage   1. Update targets vitals |
| Glancing Blow on Enemy | 1. Players Weapon is a Rifle 2. Collision detection is performed – the player must be in close quarters 3. Update targets vitals |
| Severely wound target | 1. Player measures line of site to enemy 2. Player deals close range damage to enemy 3. Update enemy vitals |
| Killed Target | 1. Player positions weapon to line of site. 2. Players weapon makes contact with target 3. Targets vitals run low enough that he dies |
| Healing as medic | 1. Receive communication over radio that soldier is injured 2. Find path to soldier 3. Follow path to soldier 4. Perform healing action until soldier is healed |
| Healed by medic | 1. Receive healing from medic |
| Dig In | 1. Dig trench |

# SUB-BEHAVIORS

Use weapon

Find line of sight

Find nearest enemy target

Path-find

Sleep

Pickup Weapon

Wrestle with Target

Find Enemy Camp

Steal Itemns from camp

Return to home camp

Find water/food

Lay down wire

Throw grenades

Hand to Hand Combat

Take Cover

Scout

Kidnap Soldier

Demolish Enemy Structure

Hide

Move to Waypoint

Team Movement

Duck

Move under direct fire

# Static Attributes (caculated on first spawn)

Physical Fitness (general indicator)

Mental Fitness (general indicator)

Base Health

Base Morale

Base Experience

Base Skills

Weapon Ability

Close Quarters Combat

First Aid

# Dynamic Attributes (changes in game)

Health

Physical Fatigue

Mental Fatigue

Stress

Morale

Cower

Experience