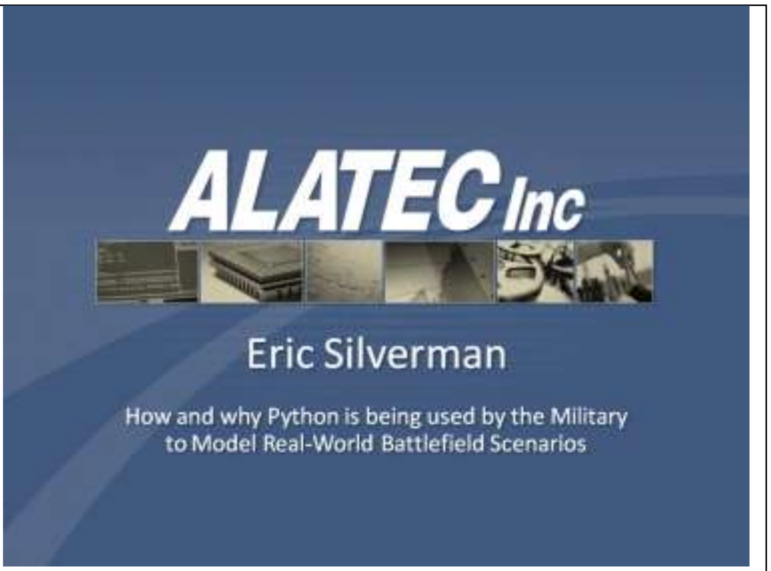
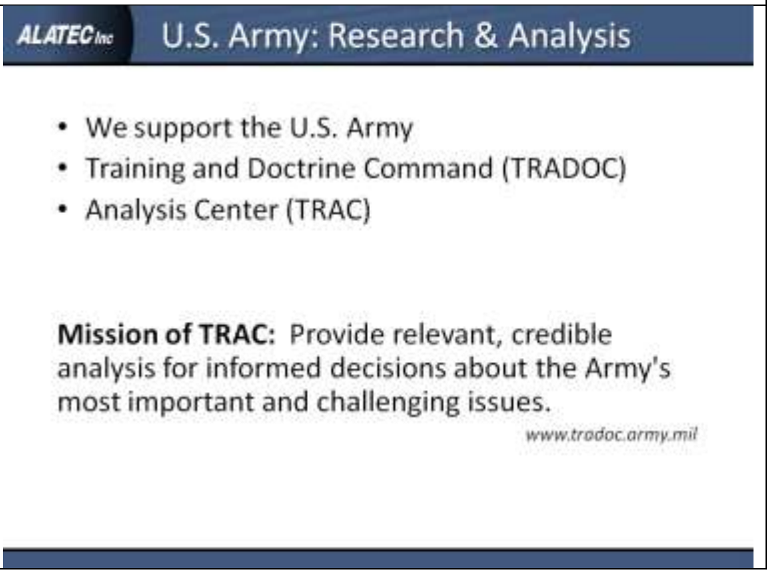
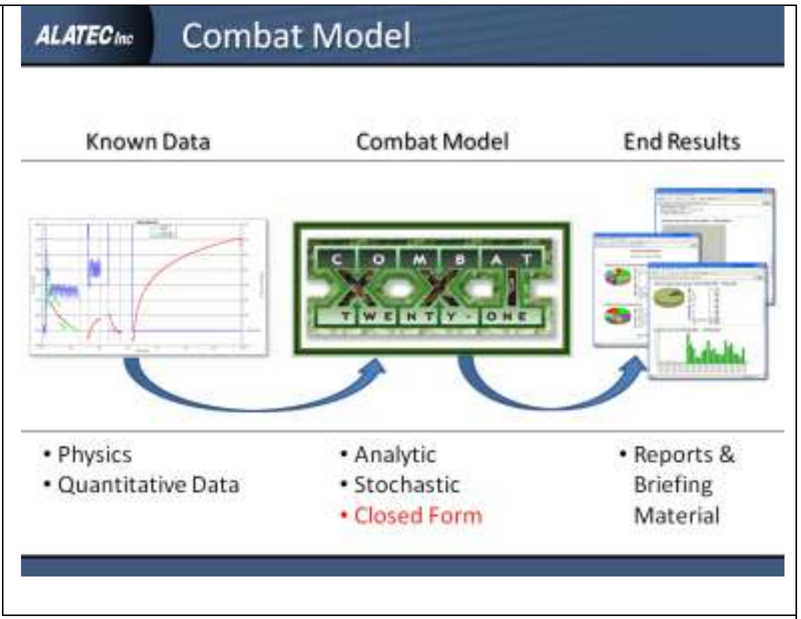
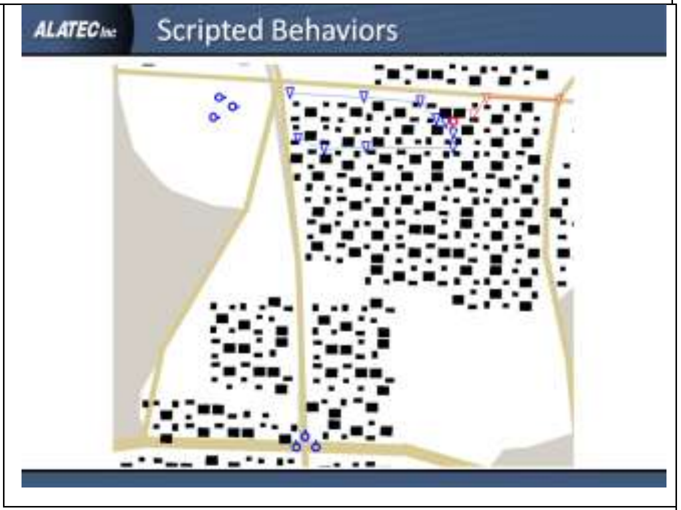
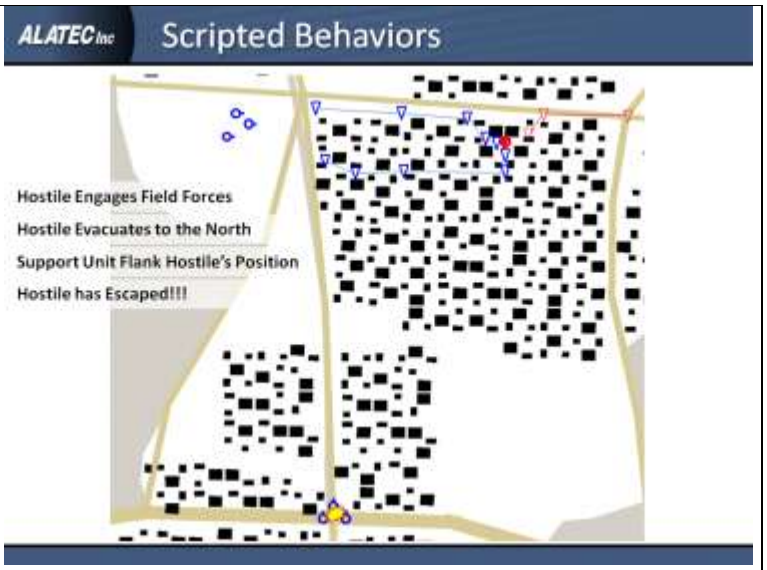
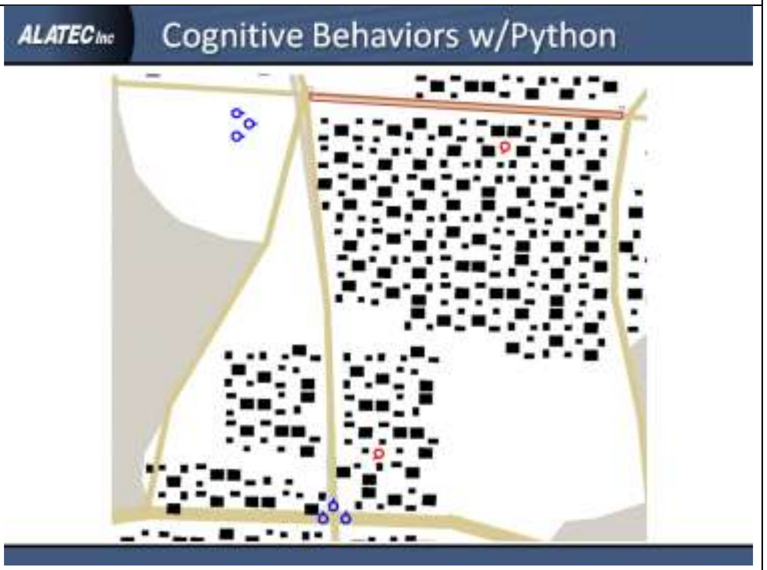
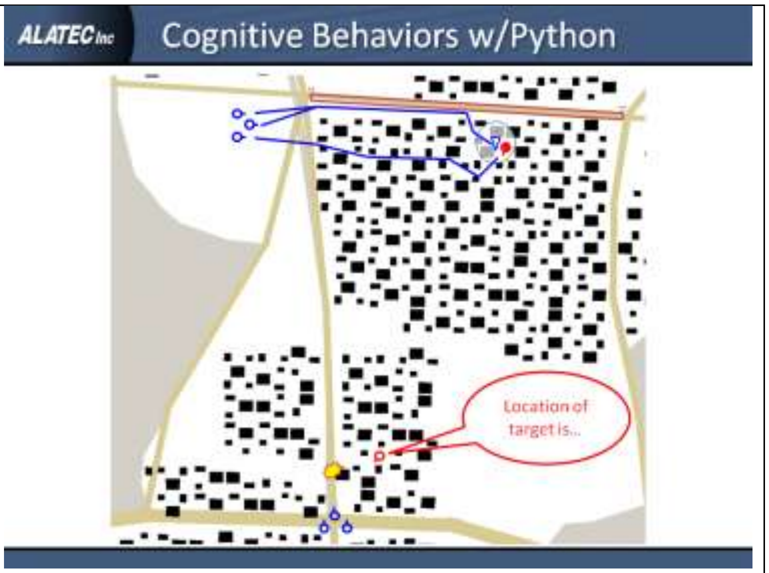
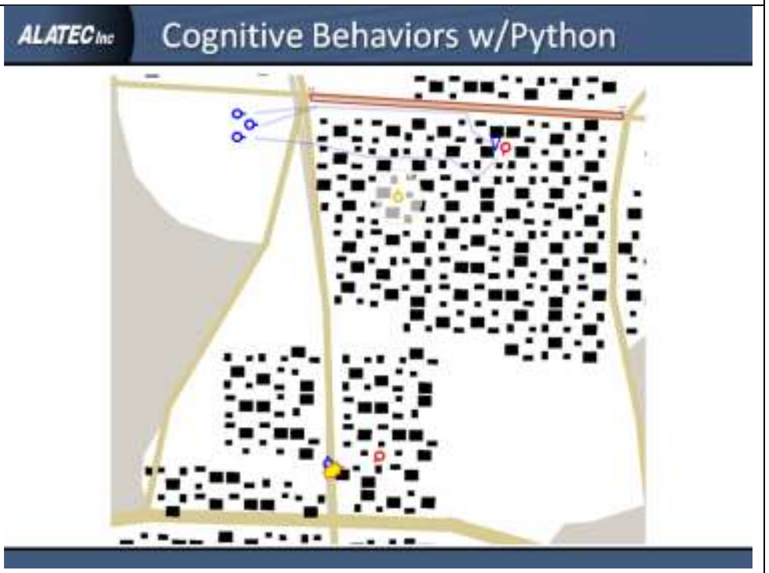
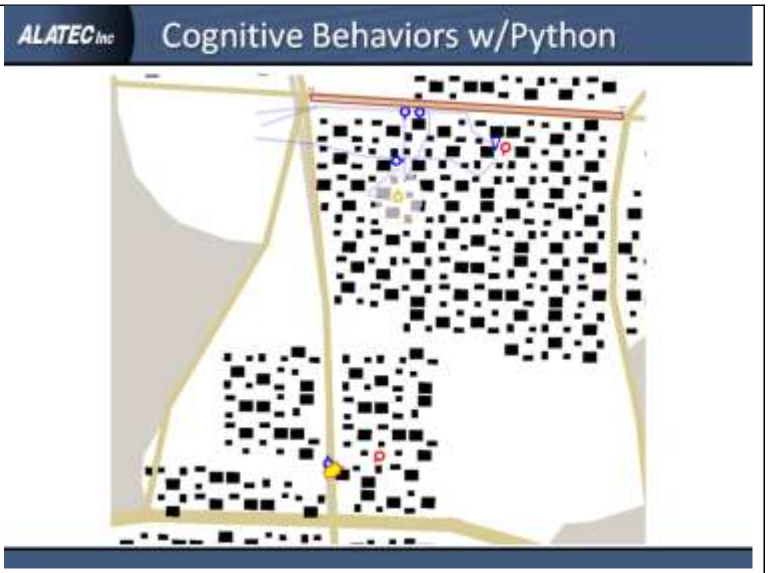
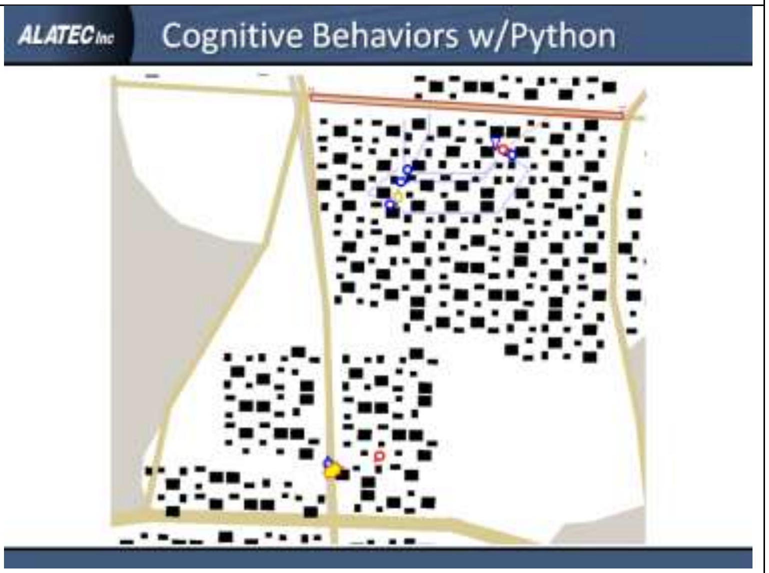
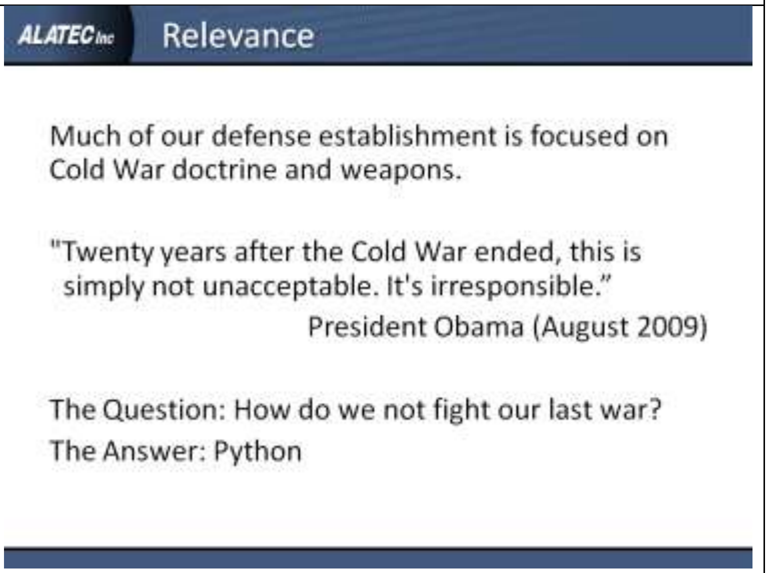
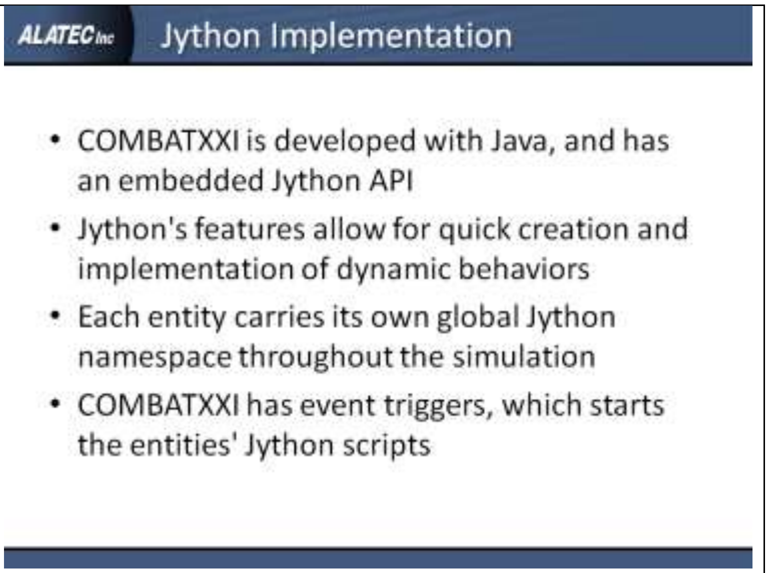
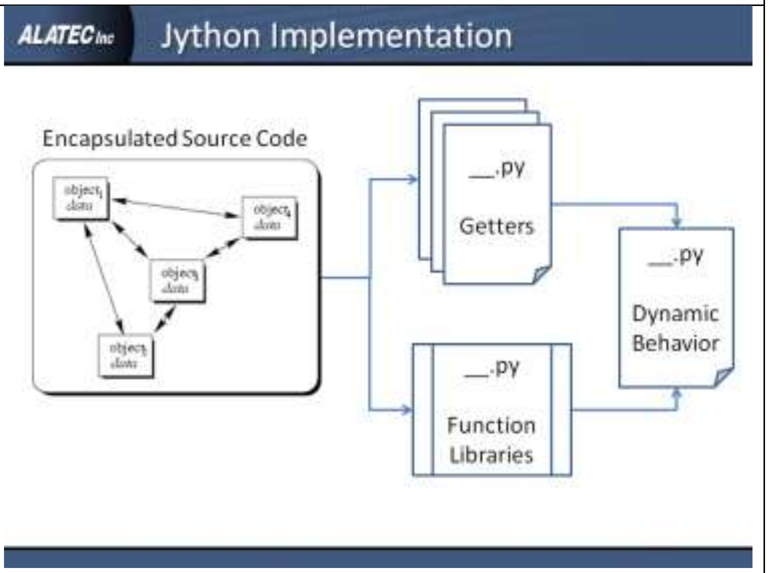
1	My name is Eric Silverman, and I am a Defense Contractor working on a project for the US Army. What I am going to discuss is the Military's use of Python for modeling battlefield scenarios. I work out in the New Mexico desert at a secure military installation on classified projects. I mention that, so that you understand that what I'm going to be showing is generalized and very watered down. I apologize ahead of time if you're underwhelmed by the examples your about to see.	 ALATEC Inc Eric Silverman How and why Python is being used by the Military to Model Real-World Battlefield Scenarios
2	I am one of many people who support the Army's Training and Doctrine Command, which is known as TRADOC. More specifically, we work for TRADOC's Analysis Center, or TRAC; which is the Army's research and analysis arm. Contrary to popular belief, the Army puts a lot of effort into research and analysis before spending millions, if not billions, of tax payers' dollars on new weapons, vehicles, and equipment. The work that is concerned with the analysis of Advanced Concepts and Requirements is done by the analyst at TRAC.	 ALATEC Inc — U.S. Army: Research & Analysis • We support the U.S. Army • Training and Doctrine Command (TRADOC) • Analysis Center (TRAC) **Mission of TRAC:** Provide relevant, credible analysis for informed decisions about the Army's most important and challenging issues. *www.tradoc.army.mil*

3	 ALATEC inc Combat Model Known Data Combat Model End Results COMBAT XXI TWENTY ONE • Physics • Quantitative Data • Analytic • Stochastic • Closed Form • Reports & Briefing Material One of the ways that the Army conducts their analysis is with computer models. Modeling a combat scenario always starts with what we know. The nature of this data is based in physics or validated quantifiable data. For example, we know that based on the size, weight, and velocity of a projectile how far it will fly within specific gravitational, atmospheric, and environmental conditions. The analysts build a scenario based on a specific battle situation, and feed in the known data. And, runs it through the model. COMBATXX is the wargaming model that we support. It uses a sophisticated stochastic system to analyze the interactions between agents on the play board. COMBATXXI is what's known as a Closed Form model. That means that all the parameters and scripts need to be developed before the scenario starts. This is where Python comes in, but we'll talk about that in a few minutes. After the model completes its run through the scenario, it outputs raw data. Military Operations Analysts (our end-users) study, evaluate, and scrutinize the gigabytes of data. From which, they create reports and briefing material that are presented to decision makers.
4	 I'm going to show you an example of a very simple battle scenario. It's very simple because of the time limitation. A complex battle scenario can't be easily explained in 30 minutes. Also, I want to warn you ahead of time: THIS IS NOT HALO!!! No cool 3-D graphics here. The power behind COMBATXXI is the stochastic analysis engine. It definitely is not graphical output. Our scenarios, without Python, use scripted – or static – behaviors. Every aspect of the scenario had to be known, planned out, and meticulously scripted. The first element in a scenario is the terrain. This is a representation of an urban terrain. There are roadways, buildings – the black blocks, and a city grid. While this is a 2-dimensional, top-down view, we do play height but do not represent it visual. In this scenario, we have placed 7 entities – or, agents. Six total good guys in 2 units, blue represents friendly forces; and 1 bad guy, red represents hostile forces. For scripted behaviors to work we need to: First, script an engagement behavior with specified weapons and targets. Second, lay down routes for all the movements, and script all the movement. And finally, test the scenario to make sure that everything is timed just right. What's going to happen is, you'll see, the red guy fire a mortar at the southern unit. Then, the good guys to the west are going to move towards the hostile position. You'll notice that there are 2 blue routes, because in the Army we always outflank the enemy. However, the bad guy is going to flee the scene, and gets away.

5	(Watch scenario play out) At this point the good guys stop because there are no scripted behavior to tell them exactly what to do and when to do it. Obviously, there are some serious shortcomings to this approach. But, rather than talking about what scripted behaviors cannot do, I'll show you an example of how we can do a lot more with the same scenario and dynamic behaviors.	 ALATEC Inc Scripted Behaviors Hostile Engages Field Forces Hostile Evacuates to the North Support Unit Flank Hostile's Position Hostile has Escaped!!!
6	Setting up a scenario for dynamic behaviors requires us to add some objects, and remove others. We start out with the same number of good guys. However, we've added a new bad guy to the south. The new bad guy is a spotter; his job is to look for his enemy the good guys, and then relay their position to the shooter. We have no preplanned routes now. However, we can designate high risk areas; such as a main thoroughfare. Generally speaking, soldiers don't like to be out in the open when in hostile territory. So, in our scenarios we designate an area that we know the blue force will avoid, and let the entities create their own path to the objective area.	 ALATEC Inc Cognitive Behaviors w/Python

7	The patrol unit will be moving northbound on the MSR (or Main Supply Route… what we civilians like to call a highway). The spotter will watch for enemy forces, and report their position, and direction of movement. When the shooter receives the location to fire on, an engagement behavior will be triggered. The shooter uses the target's location, heading, and rate of movement as parameters. Because in the real world people that are getting shot at generally like to move, we need a formula to calculate the location of the target based on its movement, and to aim at that location. This is what's known as "leading" the target. Using a device that's available to our troops, the source location of an indirect fire can be accurately determined. Indirect fire, by the way, is when the shooter and the target cannot see each other; so the shooter is firing at a location, rather than a specific target. – Notice the waypoint marker. With the location known, the support unit will create their own route to the objective area; still using a flanking maneuver.	 ALATEC Inc Cognitive Behaviors w/Python
8	…And, they will start moving out. Because we have intelligent agents on the play board, they can stop moving. If, say for example, "intel" is received about a source of information that can be interrogated while in route. This is a simple example of why Python is such a valuable tool for us. Unlike with scripted behaviors, with Python and dynamic behaviors, we have to ability break a maneuver while an agent is in route to the objective.	 ALATEC Inc Cognitive Behaviors w/Python

9	The agents, with situational awareness, can stop and create a new dynamic route to the new objective. After the interrogation is complete, they will move onto their original objective location.	
10	We also apply reactive behaviors with Python to the bad guys. So, Instead of just fleeing a tactical site, our hostile entity can maintain its position until it feels threatened. The hostile entity escapes the area only after seeing that it is outnumbered by the approaching blue forces. Of course, we can continue this scenario with how the blue force might pursue an escaping enemy. But, I think you get the idea.	
11	Why is this relevant? I believe President Obama said it best last August, when he stated that much of our defense establishment is still focused on Cold War doctrine and weapons. The President went on to say: "Twenty years after the Cold War ended, this is simply irresponsible. " Now, scripted behaviors are great if we are fighting in trench-warfare circa World War I, or even for armored battalions that are executing a Pincer maneuver in the oil fields of Kuwait. But, Irregular Warfare against enemies with IEDs that adhere to no rules of engagement; that requires a computer model that is much more flexible, much more agile, and much more adaptive. What was needed was a model with a Jython API implementation.	ALATEC Inc **Relevance** Much of our defense establishment is focused on Cold War doctrine and weapons. "Twenty years after the Cold War ended, this is simply not unacceptable. It's irresponsible." President Obama (August 2009) The Question: How do we not fight our last war? The Answer: Python

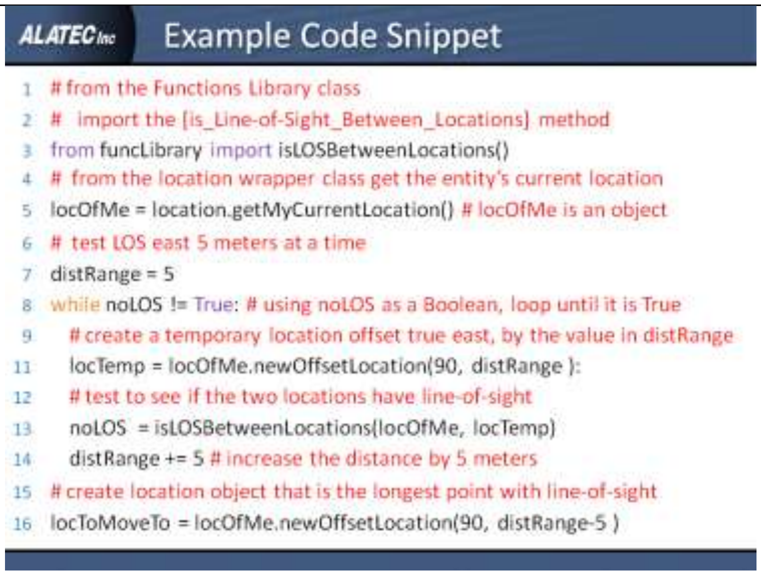
12	COMBATXXI has that embedded Jython API that allows users to control movements and actions of "entities". Entities can be almost anything - a tank, truck, soldier, or civilian. Jython's features allow for quick creation and implementation of dynamic behaviors. Each entity carries its own global Jython namespace throughout the simulation - roughly equivalent to memory and situational awareness of a real life person. COMBATXXI has events that trigger, which start the entities' Jython scripts, and causes the entity to react, or behave, in a more realistic and dynamic manner. When one entity reacts to a trigger, that entity's reaction may trigger another entity's behavior, resulting in a cascading effect across the entire scenario.	 ALATEC Inc Jython Implementation • COMBATXXI is developed with Java, and has an embedded Jython API • Jython's features allow for quick creation and implementation of dynamic behaviors • Each entity carries its own global Jython namespace throughout the simulation • COMBATXXI has event triggers, which starts the entities' Jython scripts
13	Because Jython provides un-tethered access to all of the Python, Java, and COMBATXXI source code it is very powerful - and very dangerous. Essentially, the Jython API gives the users full access to all of the Getters and Setters for all the objects in the source code. To protect the source code objects, we have created wrapper classes that provide access to the Getters, and only a select few Setters. There is also a Functions Library .py file to provide the users with useful and time saving callable method. Using the encapsulated methods, users develop the dynamic behaviors by importing the .py files with the methods that have been developed for their use.	ALATEC Inc Jython Implementation  Encapsulated Source Code object data object data object data object data __.py Getters __.py Dynamic Behavior __.py Function Libraries

14

Here is a simple example of how dynamic movement might be scripted with Python code.

We import the Functions library on line 3. Our users are non-programmers, so simple to understand and callable methods are the way to go. Admittedly, this is not the most Pythonic way to get the job done. But, in the end we would rather have something our users can understand – and with any luck, we won't offend the OOP Gods… too much.

On line 5, using the "location" wrapper class, we call the *getter* to return the entity's current location. In doing so, we'll instantiate an object from the application's source code in the Python behavior script.



Example Code Snippet

```
1  # from the Functions Library class
2  #  import the [is_Line-of-Sight_Between_Locations] method
3  from funcLibrary import isLOSBetweenLocations()
4  # from the location wrapper class get the entity's current location
5  locOfMe = location.getMyCurrentLocation() # locOfMe is an object
6  # test LOS east 5 meters at a time
7  distRange = 5
8  while noLOS != True: # using noLOS as a Boolean, loop until it is True
9     # create a temporary location offset true east, by the value in distRange
11    locTemp = locOfMe.newOffsetLocation(90, distRange ):
12    # test to see if the two locations have line-of-sight
13    noLOS  = isLOSBetweenLocations(locOfMe, locTemp)
14    distRange += 5 # increase the distance by 5 meters
15 # create location object that is the longest point with line-of-sight
16 locToMoveTo = locOfMe.newOffsetLocation(90, distRange-5 )
```

Using a few local variables, we loop and check for line of sight in 5 meter increments. By calling the is_line_of_sight_between_locations method in line 13, we test two points. When there is no line of sight, the code breaks out of the loop.

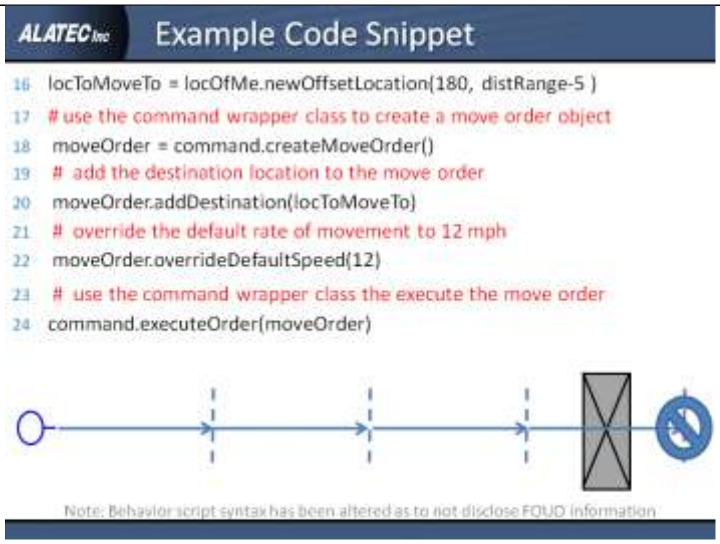
With the locOfMe object, that was istantiated in line 5, we have access to all of the object's properties and methods. One of those methods might be new_Offset_Location. By calling it, we can istantiate a new object by passing in two parameters; the direction and distance to be offsetted towards the new location.

15

With a different wrapper class called "command" we create a new object called moveOrder on line 18.

This object has its own properties and methods. In line 20, we add a location to move to, which is the object from line 16; and the speed that which the entity will move.

On line 24, we call the command wrapper class again, this time to execute the behavior.



Example Code Snippet

```
16 locToMoveTo = locOfMe.newOffsetLocation(180, distRange-5 )
17 # use the command wrapper class to create a move order object
18 moveOrder = command.createMoveOrder()
19 # add the destination location to the move order
20 moveOrder.addDestination(locToMoveTo)
21 # override the default rate of movement to 12 mph
22 moveOrder.overrideDefaultSpeed(12)
23 # use the command wrapper class the execute the move order
24 command.executeOrder(moveOrder)
```

Note: Behavior script syntax has been altered as to not disclose FOUO information

With just 13 lines of Python (not including the inline comments), we essentially created a framework for coding dynamic movement with no preplanned route. Here is an example of how this behavior would play out in a scenario with an entity that moves eastward till there is an obstacle.

We test for line of sight from the entity's current location westward in 5 meter increments.

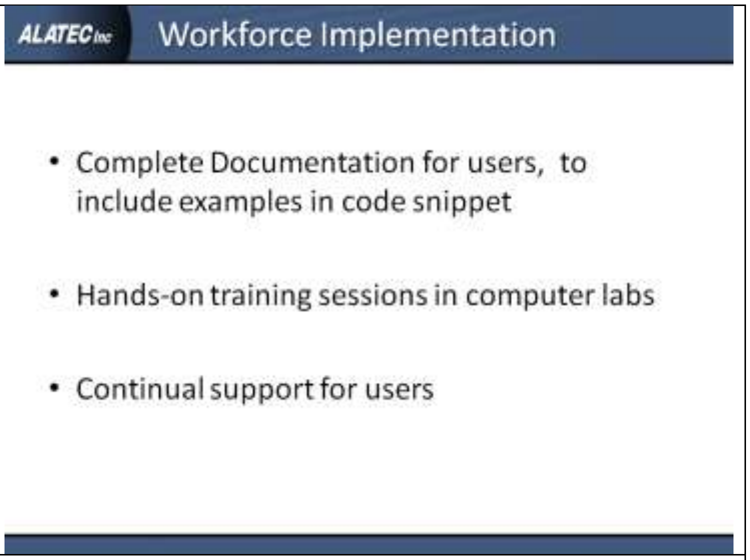
When the line of sight test returns false, the looping checks stop.

The entity uses the distance of the last successful line of sight test as the point to move to.
An order is created and executed, then the entity moves to the new location.

16 Something that has not been talked about yet is who is creating the dynamic behaviors with Python.

For the most part, they are research analysts with very little programming experience. That being the case, a lot of support needs to be provided.

To start with, a very detailed user document was authored and is maintained on an internal wiki. Because the users are not experienced programmers javadoc style documentation won't do. Instead, the user docs include full descriptions, examples on how to use the code, and even screenshots.



ALATEC Inc Workforce Implementation

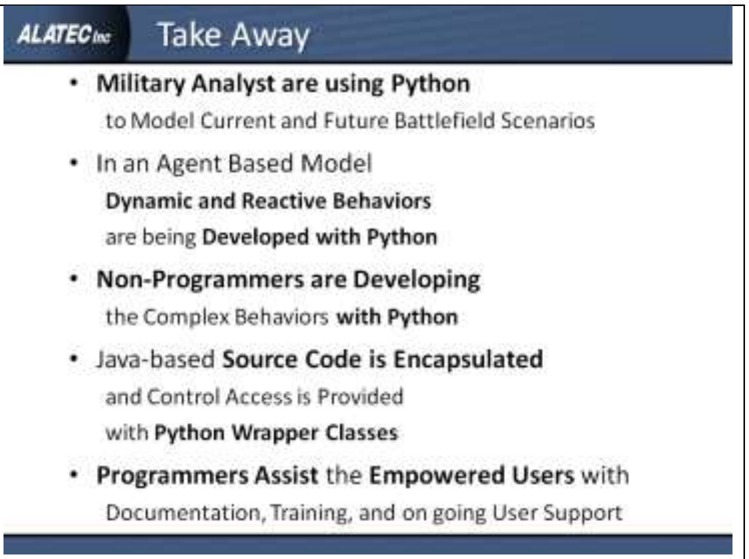
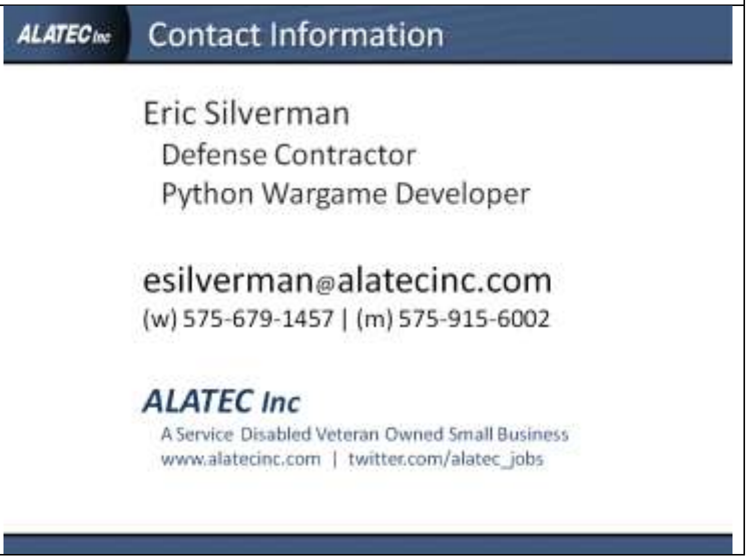
- Complete Documentation for users, to include examples in code snippet
- Hands-on training sessions in computer labs
- Continual support for users

The development group created a training program that was about 3-workdays worth of hands-on training. It started out with very simple topics that covered basic Python syntax, and then moved on to more advanced topics like using databases and class files.

This training had full support from management, which allowed the developers ample time and resources to create the training program. It also allowed the end-users time away from their daily duties in order to attend the training. This point cannot be undervalued. The actual implantation of the Jython API and creating proof-of-concept dynamic behaviors was a grass-roots effort that a couple of developers took upon themselves. However, the actual implementation into the workforce could not have happened if not for management support.

And, equally important is continual support for the users. Even if we created the most comprehensive user-docs and provided the best training possible, this effort would die in weeks if the developers did not continue to provide support. Because our user base is not made up of seasoned programmers, a lot of hand-holding is required. Sometimes that support comes as basic as how to compare a returned object to a string value and other efforts required weeks of work to help develop complex behaviors for the analyst. In these cases the goal is turn-key code that we create, but the users maintain.

The point is: a successful implementation of an open source solution at the user base level requires that we - the developers - respect the users for what their strengths are and compensate them for where their strengths are not.

17	In summary: weapons, vehicles, equipment, and tactics being used by our Military are being tested with an Agent Based Model. This Model leverages the open source and object oriented power of Python to develop Dynamic and Reactive Behaviors; which are being developed by Non-Programmers. This organization successfully implemented a Python solution because: • Control Access is provided to Java-based Source Code with Python Wrapper Classes, • Programmers Assist the Empowered Users with: Documentation, Training, and User Support, • There was upper management support.	 ALATEC Inc **Take Away** • **Military Analyst are using Python** to Model Current and Future Battlefield Scenarios • In an Agent Based Model **Dynamic and Reactive Behaviors** are being **Developed with Python** • **Non-Programmers are Developing** the Complex Behaviors **with Python** • Java-based **Source Code is Encapsulated** and Control Access is Provided with **Python Wrapper Classes** • **Programmers Assist** the **Empowered Users** with Documentation, Training, and on going User Support
18	Thank you for your time. If you have any questions, or would like so more information, please take down my contact information. I'd be happy to speak with any of you.	 ALATEC Inc **Contact Information** Eric Silverman Defense Contractor Python Wargame Developer esilverman@alatecinc.com (w) 575-679-1457 | (m) 575-915-6002 ALATEC Inc A Service Disabled Veteran Owned Small Business www.alatecinc.com | twitter.com/alatec_jobs