

Scenario Edition made easy: AEROPVS07

vsTASKER makes scenario edition easy by combining a visual paradigm with automatic C++ code generation. Developed by VirtualSim, it stands out as an open, "white-box" alternative to traditional rigid simulation tools. Instead of locking users into predefined doctrines or encrypted black-box behaviors, it offers complete freedom to precisely map out complex logic graphically.

Key Capabilities of Scenario Edition

- **State-Machine Logic:** Define custom doctrines, entity behaviors, and sensor inputs using visual diagrams.
- **Open Architecture:** Inject custom code directly into entities rather than wrestling with baffling scripts.
- **Multi-Domain Flexibility:** Build everything from simple 2D map test-beds to full 3D game-engine-driven environments.
- **Automated C++ Generation:** Turn your visual scenarios into embedded C++ code compatible with third-party graphic engines.
- **Seamless Integration:** Connect effortlessly with external platforms like VBS-IG (via CIGI) or VegaPrime without custom client-server protocols.
- Watch this introductory video to see how the software functions as a versatile white-box toolkit for engineering and programming teams:

Why It Simplifies the Process

Traditional simulation tools often hide their logic inside deep menus or demand complex scripts that require tedious debugging. vsTASKER removes these pain points by offering a visual environment where scenarios can be animated, tested, and tracked in real time. In vsTASKER 7, an automatic code updater syncs structural changes made in external

environments like Visual Studio directly back to the database, ensuring your visual logic and written code stay aligned without manual retrofitting.

Would you like to explore how vsTASKER handles a specific type of scenario, such as UAV operations, military tactical simulation, or space/satellite tracking?

vsTASKER by VirtualSim provides a visual development environment that lets you build and edit scenarios graphically without hard-coded limitations. It combines a graphical diagram editor for entity behaviors with automatic C++ code generation, giving users total control over tactical doctrines, sensors, and state-machine logic.

Key Scenario Editing Features

Unlike "black box" simulators that lock you into predefined doctrines and tasks, vsTASKER makes custom scenario creation highly accessible through its intuitive visual paradigm:

- **Visual Programming:** Build scenario logic and operational doctrines using a built-in diagram editor, allowing you to easily map out state-machine logic.
- **Entity Customization:** Assign specific behaviors, movement plans, and sensor parameters to individual entities or groups, scaling up to stress-testing 20,000 entities at once.
- **Automatic C++ Generation:** As you build your scenario, vsTASKER automatically generates the underlying C++ code. You can run simulations locally or sync your data with Visual Studio for more advanced development.
- **Flexible 2D/3D Environments:** Design on multi-layered 2D map displays, or visualize your setup and place assets directly within a 3D globe or OpenSceneGraph environment.

Platform Integrations

vsTASKER is designed to be versatile and connects readily with other simulation frameworks and visualization tools:

- **CIGI Protocol:** Connects to Visual Battlespace (VBS-IG) and other graphic/game engines to provide high-fidelity visuals without losing user-defined logic and AI.
- **VegaPrime:** Hooks directly into Presagis VegaPrime libraries to produce high-end standalone visual applications.

To see a quick overview of how vsTASKER allows developers to work without getting trapped by restrictive scripts, you can watch this introductory video:

For an overview of why vsTASKER utilizes a white-box code generation approach for programmers:

What is vsTASKER and why you should use it

You can review the full toolkit capabilities on the vsTASKER Features page or download evaluation versions from the vsTASKER Download site.

Could you tell me what **specific industry** or **use case** (e.g., military tactical simulation, air traffic control, or traffic simulation) you are working on? This will allow me to provide more targeted editing steps for your project.

Scenario configuration in **vsTASKER** involves defining the synthetic environment, setting up operational logic, and mapping entities for real-time simulation. It uses a graphical diagram paradigm to generate C++ code, allowing users to build complex behavioral models for defense, ATC, and training.

1. Map and Environment Setup

- **Terrain Data:** Import elevation models and geographical imagery (e.g., DTED, S57, or maps).
- **Weather & Physics:** Configure atmospheric conditions, wind layers, and environmental constraints that directly affect entity dynamics.
- **Sensor Logic:** Assign and calibrate synthetic sensors (Radar, Sonar) to generate detection limits and field-of-view (FOV).

2. Entity Modeling

- **Platform Definition:** Drag and drop units (air, naval, ground, space) onto the scenario map.

- **Dynamic Models:** Assign kinematic behaviors or inject custom physics models, such as those built in **MATLAB/Simulink**.
- **Behavior Tree:** Define state machines and logic diagrams that control the AI, route-finding, and reaction protocols (e.g., engage targets, evade).

3. Scenario Logic and Execution

- **Event Scheduling:** Configure trigger-based events and timelines to dictate mission sequences, threat arrivals, and radio communications.
- **Code Generation:** Use the vsTASKER diagrammatic GUI to automatically generate C++ source code, combining all logic into an executable.
- **Data Logging:** Set the runtime engine parameters to log specific metrics, trajectories, and outcomes for performance analysis.

4. Integration and Visualization

- **Interoperability:** Connect the scenario to existing systems using standards like **HLA** or **STANAG 4586**.
- **3D Viewers:** Configure the runtime output to render in 3D using viewers like **VegaPrime**, **VBS-IG**, or **Unreal Engine**.
- **HMI Builder:** Design custom operator consoles and control panels for "man-in-the-loop" simulations and training

In vsTASKER, scenario conditions are logical rules built into the entity's behavior trees or event sequences to trigger specific actions. Conditions in vsTASKER are primarily executed through the following mechanisms:

- **Logic Trees:** Defined graphically in the Activity Diagram, using **Tasks**, **Actions**, and **Choices** to dictate agent AI and conditional branching.
- **State Variables:** Monitored over time, such as fuel levels, ammo count, health, or threat detection.
- **Environment & Timing:** Triggered by environmental changes (e.g., nightfall, crossing a specific geographic coordinate) or reaching defined time milestones in the scenario.

- **Batch Analysis Terminations:** When running stress tests or trajectory optimizations, scenario conditions (such as "end simulation when the drone reaches the waypoint") automatically pause or reset the batch.

You can read more about setting up scenario nodes in the vsTASKER User Manual.

If you want, tell me:

- What **specific entity** are you trying to program? (e.g., UAV, defense radar, autonomous ground vehicle)
- What is the **conditional behavior** you need (e.g., detection, evasion, or communication logic)?

I can give you a step-by-step method to implement that in your simulation.

Scenario Edition made easy

by VirtualSim

When you need to set up a scenario and precisely define the behavior of each participant, you need a software that allows such freedom. Most of the tools on the market provide predefined tasks and doctrines that cannot be changed. You can tune some aspects but the hood is dead locked. It can still be very sexy, with stunning 3D output, but beyond the game aspect, you won't get what you needed.

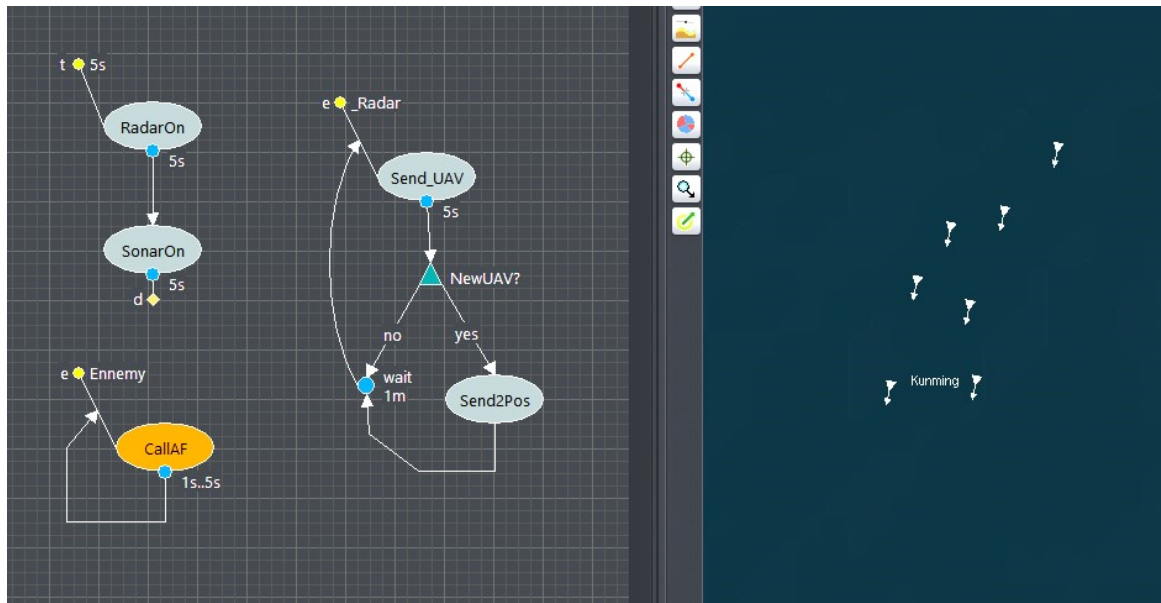
With vsTASKER, you have the ability to build your scenario the way you want. You define graphically your own doctrines based on state-machine like logics. You attach behaviors and sensors to your entities. You populate the gaming area. You specify what to do upon communications, detection assessments, signals, whatever events which can occur during the simulation, and you quickly get a realistic virtual battlefield up and running.

Taiwan Strait

Taiwan Strait Combat demo video

Ships

Three fleets of ships are defined. Two for the red force and one for the white. In the white fleet, only one ship (Kunming) has a logic, a radar and a sonar.



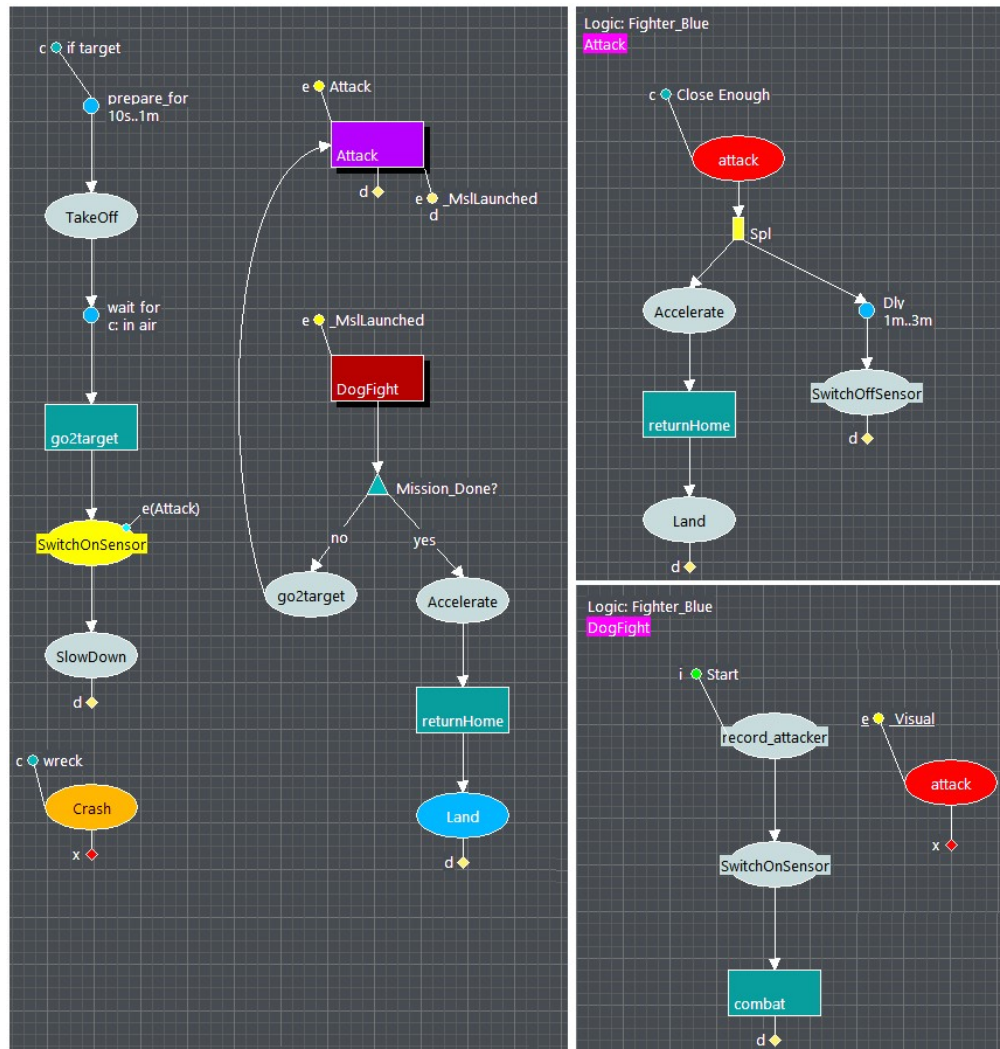
Kunming frigate logic

The two most important items of this logic are Send_UAV and Send2Pos which directs the drone to the computed position. Send_UAV consider an area and not individual detections.

Send_UAV code

Jet fighters

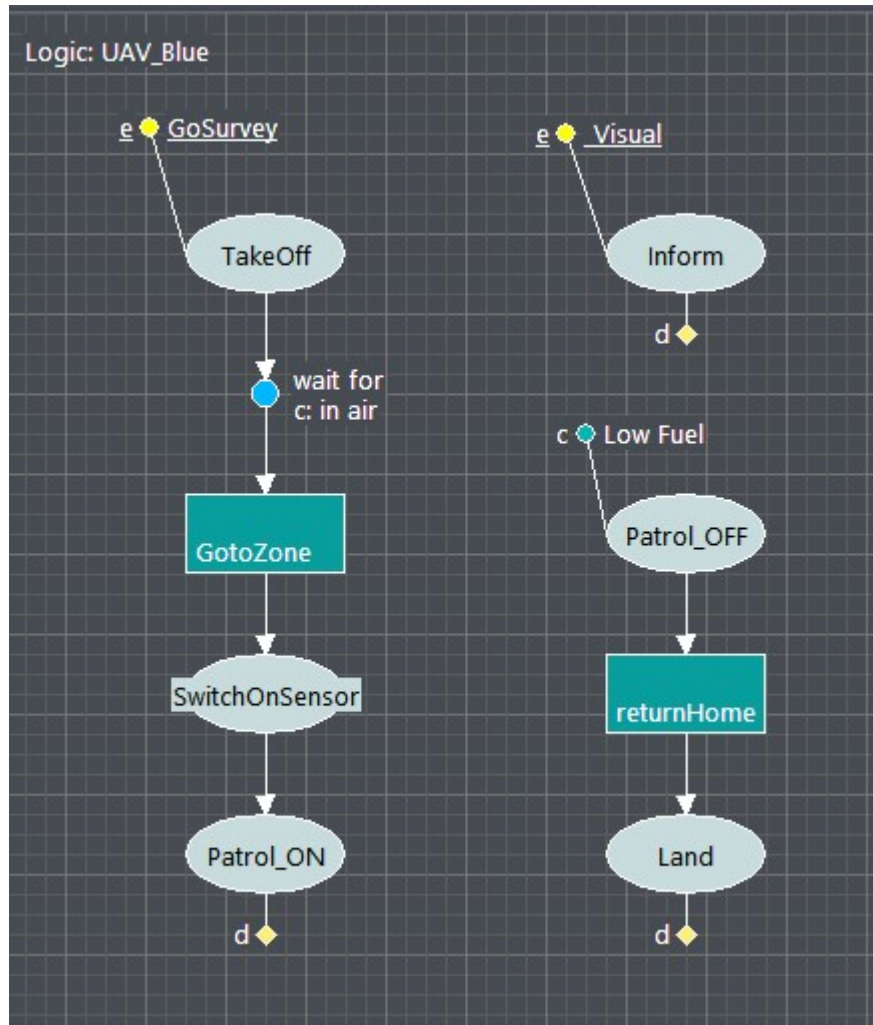
White and red jet fighters are created by air force bases upon reception of a threat by Kunming frigate (for white forces) or Yaoshan radar. All airports share the same logic. White jets have their prime mission to destroyed a given red ship. The red jets have for mission to fight and kill a specific white jet. Dogfight will be triggered when both are in visual range.



White jet fighter logic

Drones

UAV are sent by white frigate Kunming upon detection of red vessels on an area not covered by another UAV. Once launched, the drone will head towards the area and start randomly patrolling over the region. Whatever is detected by its sensor is forwarded to ship Kunming. As soon as fuel is low, the drone will return to its ship.

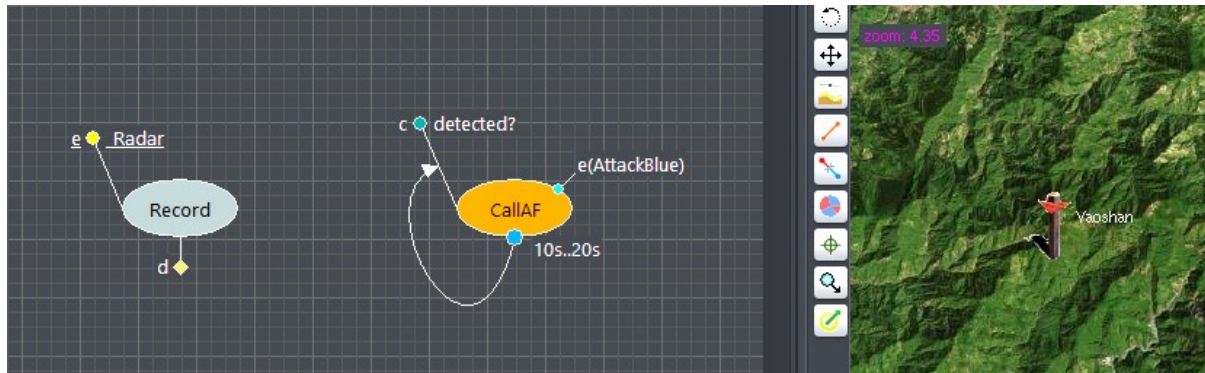


Drone main logic

Surveillance Radar

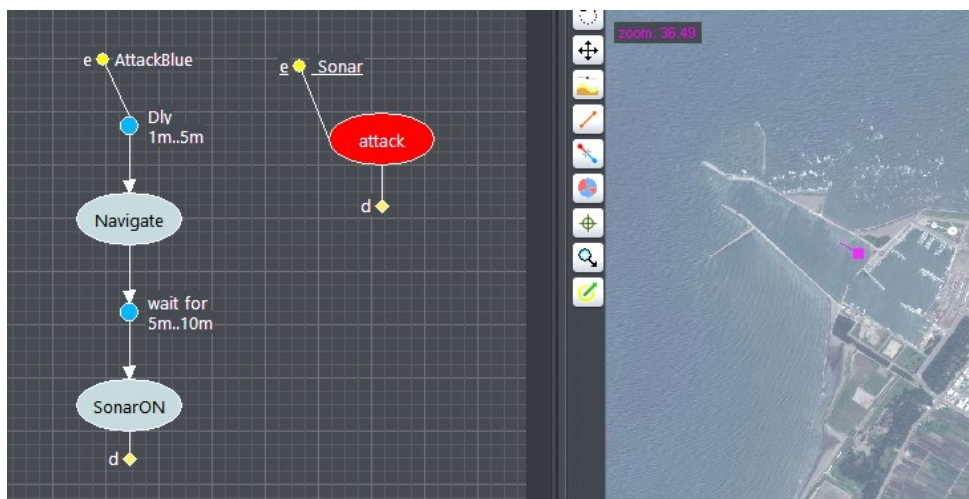
One radar has been put at the top of a mountain range, to cover the strait without shadow. The Yaoshan logic is fairly simple:

- **Record:** extract from all detections any new valid threats
- **CallIAF:** select air bases and forward all new threats



Submarines

Two submarines are stationed on coastal harbors and can receive the AttackBlue event from Yaoshan radar (see image above). When requested, they will navigate in the strait and rely on their sonar to detect enemy ships. Use of torpedoes to destroy targeted ships.



OS & HARDWARE SUPPORT

INDUSTRY APPLICATION

Note: ** Specifications are copied from manufacturer data. Please contact before buying.