



Naval Surface Warfare Center Dahlgren Division

Planner Instruction Manual

Presented by

JCORE Team

SME Prize Challenge Team

iLab

WELCOME

January 13, 2023

The Leader in Warfare Systems Development and Integration



NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION
DAHLGREN | DAM NECK



- The purpose of this challenge is for participants to develop AI/ML algorithms capable of engagement coordination and weapon pairing in a real time innovation challenge event
- To achieve this, teams will write software programs that interface with a simulation through a custom application programming interface (API) that provides the tools and information required for AI development

[Challenge.Gov](https://www.challenge.gov/?challenge=artificial-intelligence-(ai)-and-machine-learning-(ml)-algorithm-development-challenge)

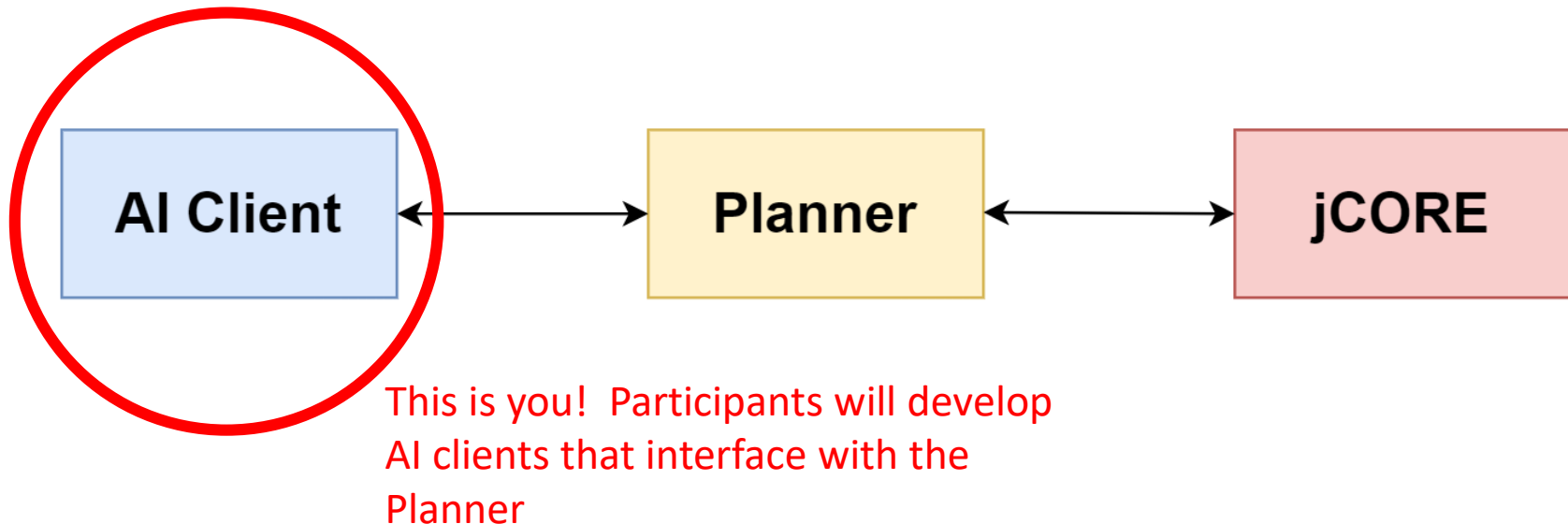
([https://www.challenge.gov/?challenge=artificial-intelligence-\(ai\)-and-machine-learning-\(ml\)-algorithm-development-challenge](https://www.challenge.gov/?challenge=artificial-intelligence-(ai)-and-machine-learning-(ml)-algorithm-development-challenge))

- The **Planner** is the AI/ML API that will interface with the simulation and was developed by the jCORE team at Naval Surface Warfare Center Dahlgren Division
- The simulation is the the **Joint Cognitive Operational Research Environment (jCORE)** developed by Metateq, Inc.
- jCORE is a medium fidelity modeling and simulation environment that is flexible and structured as a commercial video game



jCORE Information: [MetaTeq, Inc. | Home](https://metateq.com/) (https://metateq.com/)

- The Planner sits between jCORE and AI clients and provides an easy-to-use interface not specific to any software language
- The Planner will handle state information, client actions, scoring, and configuration/controlling of the simulation.
- The Planner environment and clients should be run on the same windows machine



- jCORE will provide live naval combat scenarios that AI/ML clients will use as their simulation environment
- Developed AIs will learn and be tested by “playing” these scenarios, not by digesting pre-processed data
- AI’s will control the engagement actions of ***ships*** and attempt to defeat a raid of **enemy missiles**
- The simulation has a realistic physics engine and is based on real-time with options for accelerated execution

- The jCORE scenarios will be limited to the following types of assets:
 - ***Ships***: Surface vessels that can launch multiple types of missiles at incoming enemy threats
 - ***Friendly Missiles***: Missiles launched from ships to destroy ***enemy Missiles***
 - ***Enemy Missiles***: Missiles with the goal of targeting and destroying ***ships***
- The client simulation actions will consist of engaging incoming enemy threats with certain friendly kinetic weapons
- There are no enemy ships

- There are two types of ships:
 - **Galleon**: A medium sized modern naval vessel with two missile systems
 - **High Value Unit (HVV) Galleon**: A Galleon containing precious cargo and/or critical technologies that has a higher penalty for successful enemy actions against it
- All ships will have the same capabilities, weapon types, and health
- Ships can only fire defensive missiles. They cannot move or perform any other actions
- Ships have a health of **4**, meaning 4 hits from enemy missile will result in total ship destruction
- All data will be referenced to a single ship, which can be determined by its Cartesian values in the state information being zero vectors

- Scenarios will always have a reference ship that is shown in the state messages as below
- The reference ship relays track data to the friendly ships
- All friendly ships have x, y, z (East, North, up) values relative to the reference ship
- **Reference ships:**
 - Are invincible and invisible to the enemy
 - Cannot shoot or perform any actions
 - Have a health of -1 (as it is not an active actor)
 - Are always reported in the state messages as seen below

```
1: Galleon_REFERENCE_SHIP
2: false
3: -1
4: 0.0
5: 0.0
6: 0.0
7: [25.0, -85.0, 0.0]
8: []
```

```
message AssetPb {
    string AssetName = 1;           // Name of the assets (AssetName in ShipActionPb)
    bool isHMU = 2;                 // Whether or not this asset is a high value unit
    int32 health = 3;               // Total health of the asset
    double PositionX = 4;           // Relative position East (meters)
    double PositionY = 5;           // Relative position North (meters)
    double PositionZ = 6;           // Relative position Up (meters)
    repeated double Lle = 7;        // Latitude, longitude, elevation of asset
    repeated WeaponPb weapons = 8;  // State of asset's deployers
}
```


- ***Cannon:*** A high speed missile system
 - Speed: 972m/s
- ***Chainshot:*** A low speed missile system
 - Speed: 343m/s
- Both missile systems can destroy an enemy missile with one successful hit
- Both missile systems have single track targeting, meaning once launched will only go for its original track

- There are 4 types of enemy missiles:
 - **Buccaneer V1**: Speed = 686m/s | Cruising Altitude = 1000m
 - **Buccaneer V2**: Speed = 686m/s | Cruising Altitude = 20m
 - **Privateer V1**: Speed = 292m/s | Cruising Altitude = 1000m
 - **Privateer V2**: Speed = 292m/s | Cruising Altitude = 20m
- All enemy missiles do the same damage to friendly ships (4 hits to destroy a ship)
- Any friendly missile has the “capacity” to defeat any enemy missile depending on the situation
- Enemy missiles cannot target friendly missiles, only friendly ships

- All provided and procedurally generated scenarios will have one reference ship, a maximum of **5 friendly acting ships** (max of one high value ship), and a maximum of **30 enemy threats**
- Friendly ship missile inventory has no given maximum or minimum
- **Tested scenarios will also be within these bounds**

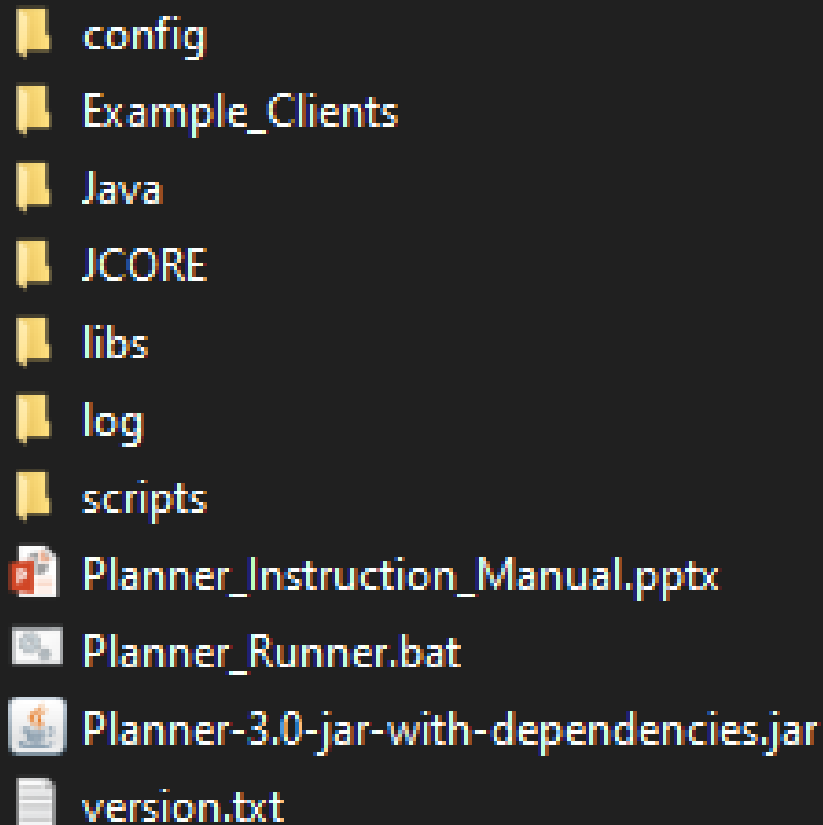
- Developed AIs will make decisions about threat engagement and those decisions will be translated into simulation actions by ***The Planner***



- AI clients are permitted one action, per ship, per time interval
- Actions/decisions will consist of:
 - What friendly ship is acting
 - What the friendly ship is acting with (type of weapon/system)
 - Who the friendly ship is acting against (which enemy threat)

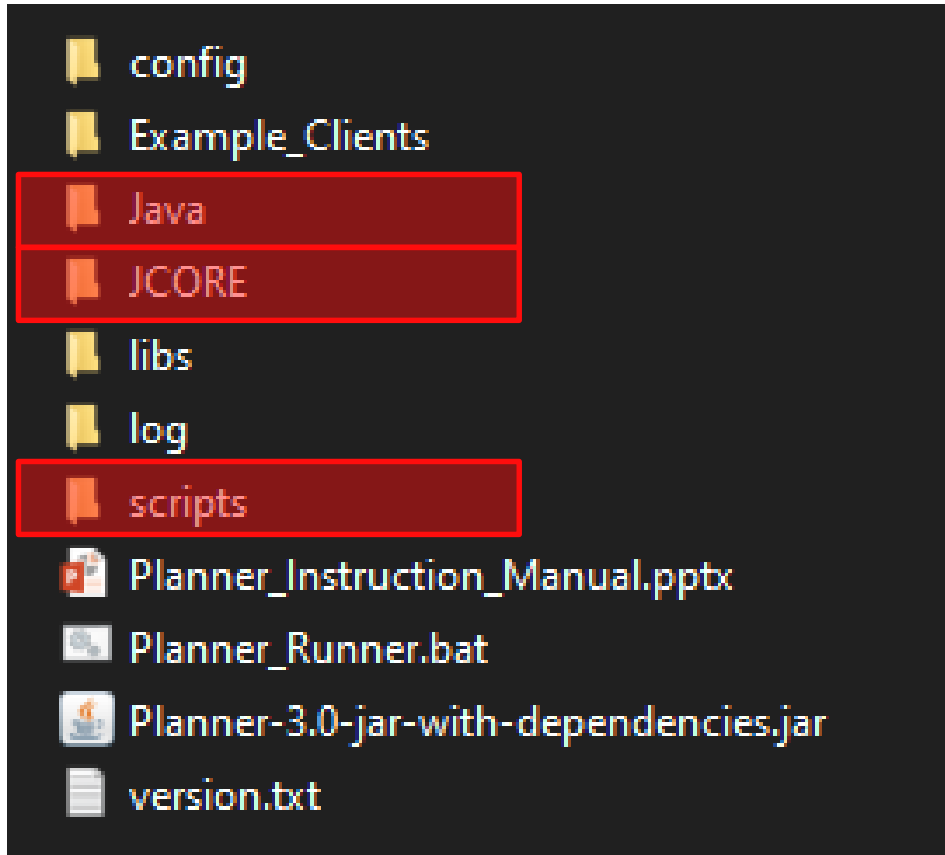


- The file ***Planner_Environment.zip*** contains all the required data and applications for this challenge



- config
- Example_Clients
- Java
- JCORE
- libs
- log
- scripts
- Planner_Instruction_Manual.pptx
- Planner_Runner.bat
- Planner-3.0-jar-with-dependencies.jar
- version.txt

- The files highlighted below should not and do not need to be edited



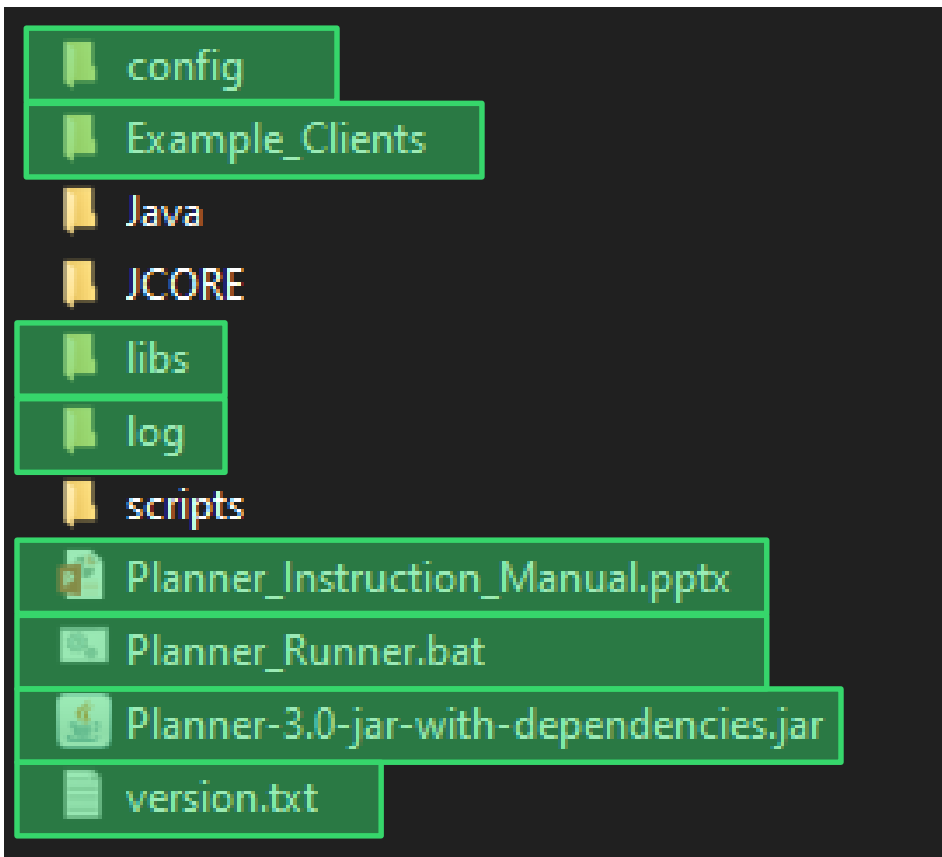
Java: contains jdk required for java client

JCORE: contains JCORE simulation

scripts: contains commands to properly run and terminate applications

Editing these files could result in an unusable environment

- The files you should be familiar with are highlighted in green below



config: contains configuration settings for the Planner

Example_Clients: contains starter clients

libs: contains proto definitions and required libraries for custom clients

log: location for Planner log files

Planner_Instruction_Manual: this

Planner_Runner.bat: Runs the Planner

Planner-3.0: Planner executable

version.txt: contains build information and patch notes for Planner

- To set up the Planner, the only file that needs to be edited is in config > **metricsApp.config**
- Replace <YOUR_PATH> to match that of where you installed the environment on the fields seen below
- Ensure everything is working by running the Planner in step mode with an example client

```
# Location of JCORE executable
jcore.default.folder=C:/<YOUR_PATH>/Planner_Environment/JCORE/JCORE

# this is the default scenario for the planner.
jcore.default.scenario=C:/<YOUR_PATH>/Planner_Environment/JCORE/JCORE/Data/PrizeChallengeData/AI_training_template.json

# Path to script to stop jcore related processes
jcore.stopPath = C:/<YOUR_PATH>/Planner_Environment/scripts/jcore-stop.bat
```

- The Planner is used through a GUI that has data fields to control the different modes, training data, and more



The screenshot displays the 'Planner' tab of a graphical user interface. At the top, there are two tabs: 'Planner' (selected) and 'Scenario Generator'. Below the tabs, the text 'POWERED BY JCORE' is visible, with 'JCORE' in a large, bold font and 'Joint Cognitive Operational Research Environment' in smaller text below it. The main area contains several input fields and buttons. On the left, there are labels for 'Jcore Version', 'Scenario(s)', '# Runs', 'Step (int)', 'Seed (int)', 'Time Mode', and 'Scenario Gen'. Each label is followed by an input field. To the right of these fields are buttons for 'File...' and 'Folder...'. At the bottom left, there is a 'Run.' button. At the bottom right, there is a 'Stop!' button. Below the input fields, there is a 'Console' section with a large text area for output. The background is dark gray.

Planner Scenario Generator

POWERED BY
JCORE
Joint Cognitive Operational Research Environment

Jcore Version C:/devWork/Prize_Challenge/Planner_Enviror File...

Scenario(s) C:/devWork/Prize_Challenge/Planner_Enviror File... Folder...

Runs 1

Step (int) 25

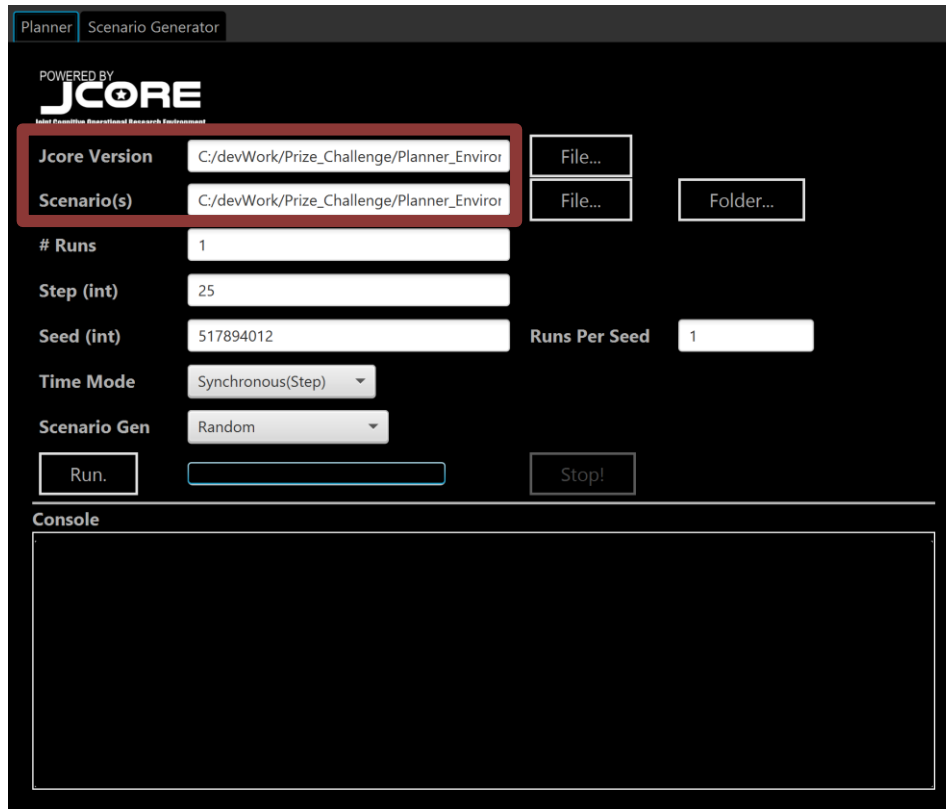
Seed (int) 517894012 Runs Per Seed 1

Time Mode Synchronous(Step) ▼

Scenario Gen Random ▼

Run. Stop!

Console



The screenshot shows the 'Planner Scenario Generator' window. It features a dark theme with white text. At the top left, it says 'POWERED BY JCORE'. The main area contains several input fields and buttons. A red rectangle highlights the 'Jcore Version' and 'Scenario(s)' fields, both containing the path 'C:/devWork/Prize_Challenge/Planner_Enviror'. To the right of these fields are 'File...' and 'Folder...' buttons. Below these are fields for '# Runs' (1), 'Step (int)' (25), 'Seed (int)' (517894012), and 'Runs Per Seed' (1). There are also dropdown menus for 'Time Mode' (set to 'Synchronous(Step)') and 'Scenario Gen' (set to 'Random'). At the bottom left is a 'Run.' button, and at the bottom right is a 'Stop!' button. A 'Console' area is at the very bottom, currently empty.

Planner Scenario Generator

POWERED BY
JCORE
Joint Expeditionary Operational Research Environment

Jcore Version C:/devWork/Prize_Challenge/Planner_Enviror File...

Scenario(s) C:/devWork/Prize_Challenge/Planner_Enviror File... Folder...

Runs 1

Step (int) 25

Seed (int) 517894012 Runs Per Seed 1

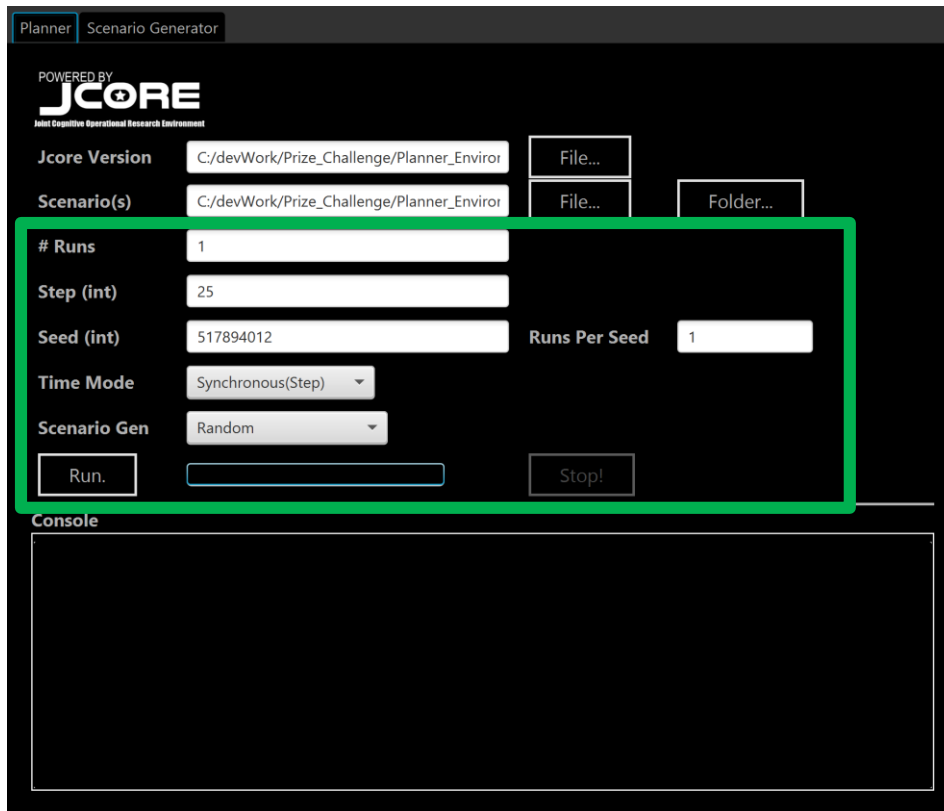
Time Mode Synchronous(Step) ▼

Scenario Gen Random ▼

Run. Stop!

Console

- *Jcore Version* and *Scenario(s)* are related to the simulation configuration and should not be edited unless otherwise instructed.



The screenshot shows the 'Planner' tab of the 'Scenario Generator' interface. The 'POWERED BY JCORE' logo is at the top left. Below it, the 'Jcore Version' and 'Scenario(s)' fields are set to 'C:/devWork/Prize_Challenge/Planner_Enviror', each with a 'File...' button. The configuration section, highlighted by a green box, includes: '# Runs' (1), 'Step (int)' (25), 'Seed (int)' (517894012), 'Runs Per Seed' (1), 'Time Mode' (Synchronous(Step)), and 'Scenario Gen' (Random). At the bottom of this section are 'Run.' and 'Stop!' buttons. Below the configuration section is a 'Console' area with a large empty text box.

- ***# Runs, Step, Seed, Runs Per Seed, Time Mode, and Scenario Gen*** are free to be edited and control “how” the simulation is run

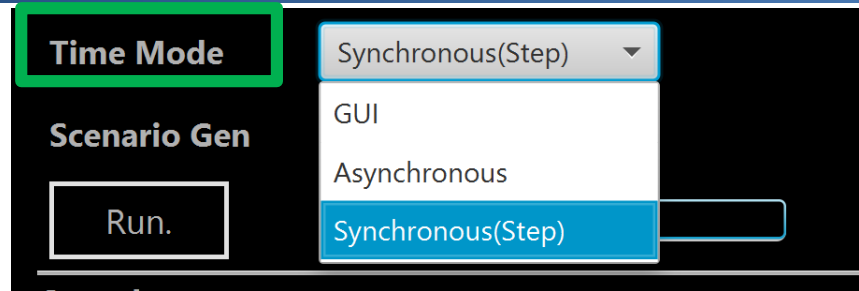
Scenario(s)	C:/devWork/Prize_Challenge/Planner_Test_Bi	File...	Folder...
# Runs	1		
Step (int)	25		
Seed (int)	517894012		

- Each scenario has a maximum real-time length of 5 minutes
- **# Runs** controls how many scenarios are selected/generated and run
 - Multiple runs on scenarios labeled with “CONST” will always be the same scenario
 - Multiple runs on scenarios labeled “Random” will use **Seed** and **Runs Per Seed**

# Runs	1
Step (int)	25
Seed (int)	517894012
Time Mode	Synchronous(Step)

Runs Per Seed	1
---------------	---

- **Step** controls the number of discrete time steps between simulation states in jCORE
 - A step of **25** relates to 1 in-game second (300 steps to complete run)
 - A step of **50** relates to 2 in-game seconds (150 steps to complete run)
 - The **Step** field only has an impact during runs where **Synchronous (Step)** is selected for the **Time Mode**
 - **Tested scenarios will be in real-time, i.e. step of 25, which is 1 second between state messages**
- **Seed** is the starting seed for the random scenario generator
- **Runs Per Seed** is the number of scenarios run before the seed changes
- Runs performed with the same seed will be identical scenarios, and successive seeds are produced randomly from the previous seed. Thus, identical sets of runs can be achieved



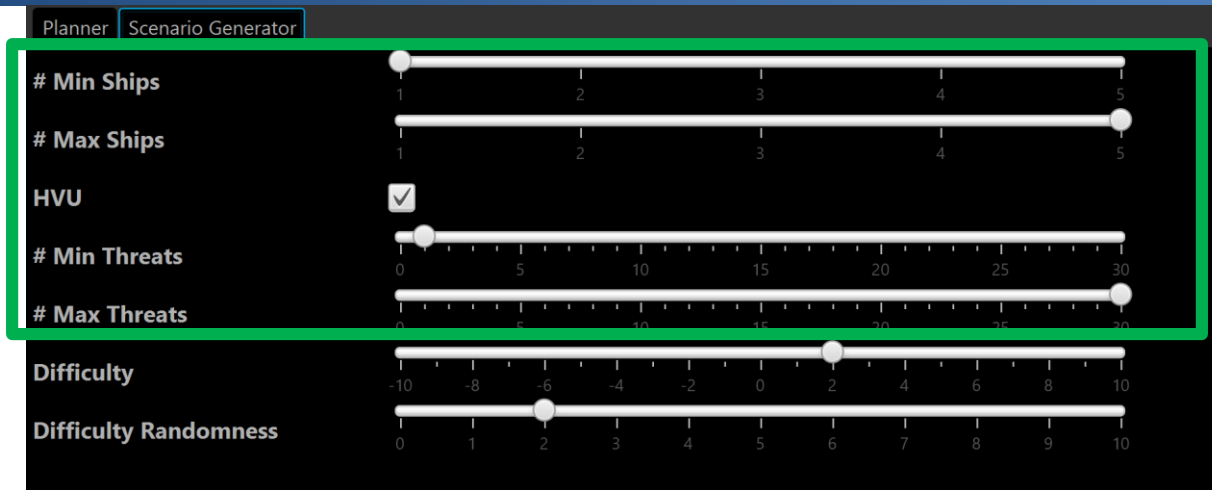
- ***Time Mode*** controls the method the simulation is run
 - ***GUI*** runs the scenario in real-time (1 second between state messages) and opens the jCORE GUI where the simulation can be visually observed. **GUI** mode is asynchronous to the client
 - ***Asynchronous*** runs the simulation without a GUI (headless) and does not wait for the client to progress through the scenario
 - ***Synchronous (Step)*** runs the simulation and waits for client inputs/commands after each time time-step



- ***Scenario Gen*** selects the type of scenario to run
- Possible options may include:
 - Randomly generated scenarios (Labeled *Random*)
 - Various fixed/constant scenarios (Labeled *CONST*)
- The setting selected is what will be used until all runs set in ***# Runs*** have been completed

- Scenario Gen will have four initial options:
 - **Random:** Random scenarios controlled by seed and the Scenario Generator tab
 - **CONST_min_raid:** A scenario with one ship and one enemy missile
 - **CONST_4_threats:** A scenario with two ships and four enemy missiles
 - **CONST_max_raid:** A scenario with five ships and thirty enemy missiles
- The “CONST” scenarios are useful for testing clients and ensuring their operation and data handling is correct

Scenario Gen Configurations



Planner Scenario Generator

Min Ships

Max Ships

HVV ☒

Min Threats

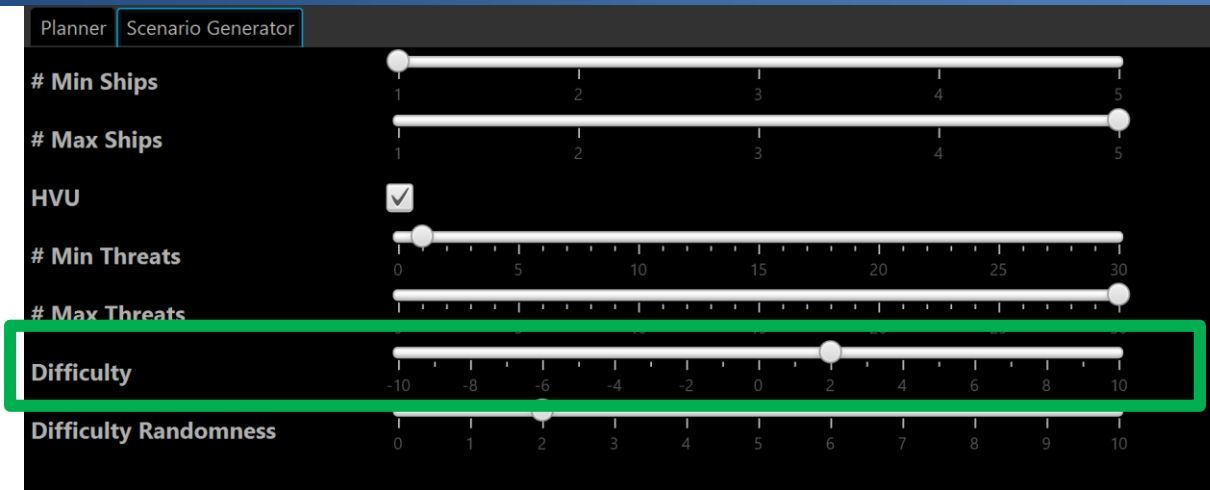
Max Threats

Difficulty

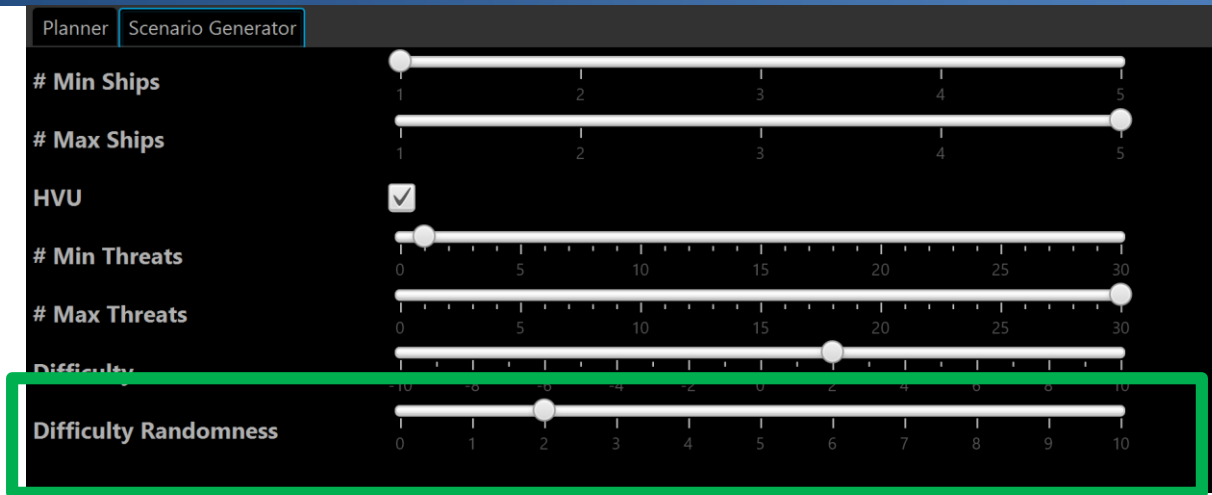
Difficulty Randomness

- If **Random** is selected under **Scenario Gen** the following sliders and fields will be used to scale the randomness
- The interval for random generation of ships and enemy threats can be set by manually setting the interval with the first four sliders
- If the maximum slider is less than the minimum, the minimum slider value will be used without randomization for that field
- If **HVV** is selected, there will be a 50% change of having a high value asset in the scenario

Scenario Gen Difficulty



- **Difficulty** is based on how many missiles each ship needs to complete the scenario unscathed
- Generated inventories will be equal among friendly ships
- A **Difficulty** of 0 means that the friendly ships will have enough missiles per ship to defeat every enemy threat
- A **Difficulty** of -1 means each ship will have one more missile than needed
- A **Difficulty** of 1 means each ship will have one less missile than needed



Planner Scenario Generator

Min Ships: 1 to 5 (slider at 1)

Max Ships: 1 to 5 (slider at 5)

HVU: ☒

Min Threats: 0 to 30 (slider at 0)

Max Threats: 0 to 30 (slider at 30)

Difficulty: -10 to 10 (slider at 0)

Difficulty Randomness: -10 to 10 (slider at 2)

- **Difficulty Randomness** allows for runs with different levels of difficulty
- Assuming **Difficulty** is 0, setting the randomness to 1 means there is a chance you could have one more or one less than the required inventory per ship to defeat the raid unscathed

- Ships = 5
- Threats = 30
- Required Per Ship Inventory = 6
- Difficulty = 0
- Difficulty Randomness = 0
- **Resulting Per Ship Inventory = 6**
- **Total Friendly Missiles = 30**

- Ships = 5
- Threats = 30
- Required Per Ship Inventory = 6
- Difficulty = 1
- Difficulty Randomness = 0
- **Resulting Per Ship Inventory = 5**
- **Total Friendly Missiles = 25**

- Ships = 5
 - Threats = 30
 - Required Per Ship Inventory = 6
 - Difficulty = 1
 - Difficulty Randomness = 1
 - **Resulting Per Ship Inventory Range = [4, 6]**
 - **Total Friendly Missiles Range = [20, 30]**
-
- Ships = 5
 - Threats = 30
 - Required Per Ship Inventory = 6
 - Difficulty = 0
 - Difficulty Randomness = 2
 - **Resulting Per Ship Inventory Range = [4, 8]**
 - **Total Friendly Missiles Range = [20, 40]**

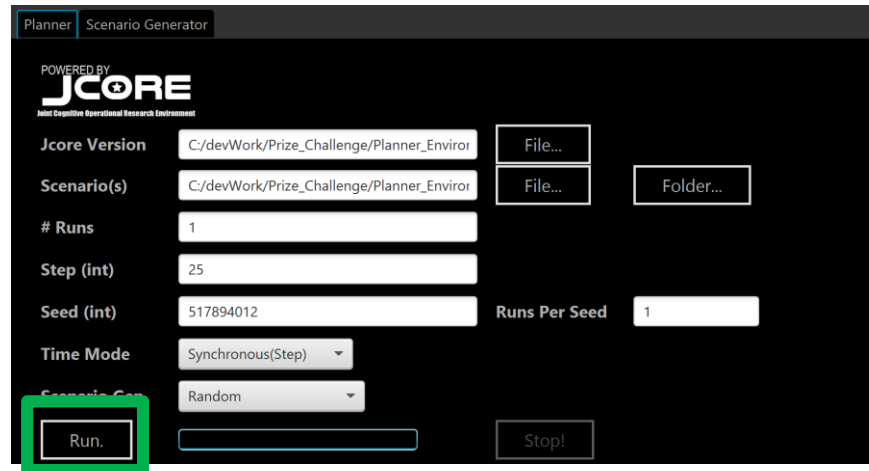
WARNING

Learning AIs should be exposed to a variety of situations. The **Planner** and **jCORE** does NOT ensure that generated scenarios are what a specific AI should be learning. Scenario Generation is a tool and should be adjusted as needed. Tested scenarios can and will be different but will have the same maximum state space and action space.

WARNING

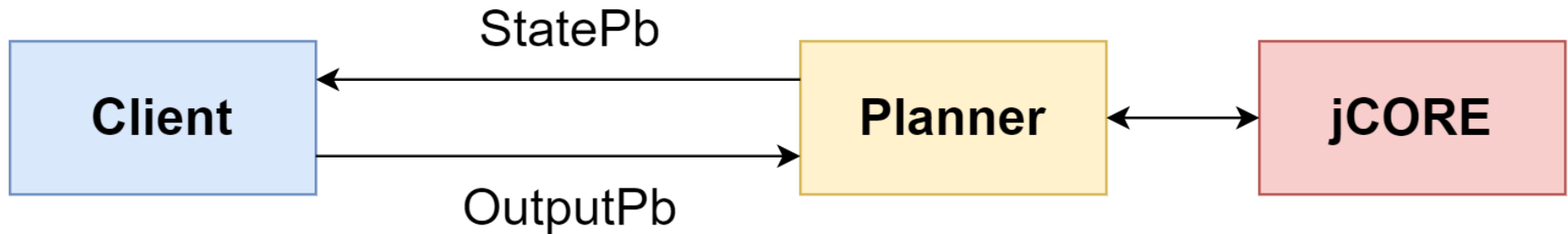
The scenario generator could generate a tested scenario. A tested scenario could also not be possible in the scenario generator. However, as stated, it is **guaranteed** that every tested scenario **will** be 5 minutes long, have a reference ship, a maximum of 5 friendly ships (including a maximum of 1 HVU), and a maximum of 30 enemy missiles

Starting and Stopping Execution



- With the settings set as desired, click **Run** to start a set of runs
- Clicking **Stop** will halt execution after the run has finished
- Closing the **Planner** by killing the process will also stop communication with the client without notifying the client that the scenario is “over”, but is acceptable to do if you wish for the run to be ended early

- Starter code in Python and Java will be provided and well documented/commented
- Starter code is not required to be used, but is **highly recommended**, at least as a reference
- Planner Client Requirements Include:
 - ZeroMQ for transport layer communication
 - Publisher Port on **tcp://127.0.0.1:8885**
 - Subscriber Port on **tcp://127.0.0.1:8886**
 - Google Protocol Buffers
 - Access to the Planner/Client protocol buffer messages (provided with example clients)



- Every second and/or time step the Planner will post a StatePb message (Pb stands for protocol buffer)
- In **Synchronous(Step)** mode, the Planner will wait for an OutputPb back to continue the simulation
- If not in **Synchronous(Step)** mode, the Planner will continue to send StatePb messages and parse OutputPbs but execution is asynchronous with clients

- An example of a protobuf message exchanged between the Planner and client is seen below
- The protobuf message definitions can be found in the “libs” folder of the provided environment in ***PlannerProto.proto***
- Enhanced understanding of Protocol Buffers can be found at [Overview | Protocol Buffers | Google Developers](https://developers.google.com/protocol-buffers/docs/overview) (<https://developers.google.com/protocol-buffers/docs/overview>)

```
message TrackPb {  
    int32 TrackId = 1;           // Track's unique id (TargetId in ShipActionPb)  
    string ThreatId = 2;         // Track's name  
    string ThreatRelationship = 3; // Friendly, Hostile, Neutral  
    repeated double Llc = 4;     // Latitude, Longitude, Elevation  
    double PositionX = 5;        // Relative position East (meters)  
    double PositionY = 6;        // Relative position North (meters)  
    double PositionZ = 7;        // Relative position Up (meters)  
    double VelocityX = 8;        // Absolute velocity East (meters/sec)  
    double VelocityY = 9;        // Absolute velocity North (meters/sec)  
    double VelocityZ = 10;       // Absolute velocity Up (meters/sec)  
}
```

- Below is the message format the client uses to relay actions to the **Planner**. *ShipActionPb actions* is a repeated field, meaning multiple of them can exist in a single OutputPb
- Each OutputPb is allowed one ShipActionPb per ship, meaning each ship gets one action per set of actions relayed to the **Planner** in an OutputPb

```
message OutputPb {  
    repeated ShipActionPb actions = 1;  
}
```

- Below is the format for a ShipActionPb message
- The fields are:
 - TargetId: A unique integer for an enemy threat
 - AssetName: The name of the ship to perform the action
 - Weapon: The type of weapon for this action
- All of the required data to perform an action is given within the StatePb message
- As stated reference the ***PlannerProto.proto*** file in the provided environment to see definitions for all the protobuf messages

```
message ShipActionPb {  
    int32 TargetId = 1;           // Track's unique Id  
    string AssetName = 2;        // Asset engaging this track  
    string weapon = 3;           // Weapon to deploy  
}
```

- As stated, starter code is provided in Java and Python
- Participants are free to write their own clients from scratch or simply edit the starter clients
- The Python and Java starter clients have similar structures
- **It is highly recommended** to begin with the provided starter code and expand/edit from there

- The Java starter client is a [Maven](#) project with 5 classes: *Main*, *AiManager*, *Publisher*, *Subscriber*, and *SuscribeTo*
- Only *AiManager* needs to be edited under normal circumstances
- There are special methods in *AiManager* that are called when the message listed in its annotation is received by the client
- These methods are good areas to directly implement AIs or call a more complex class

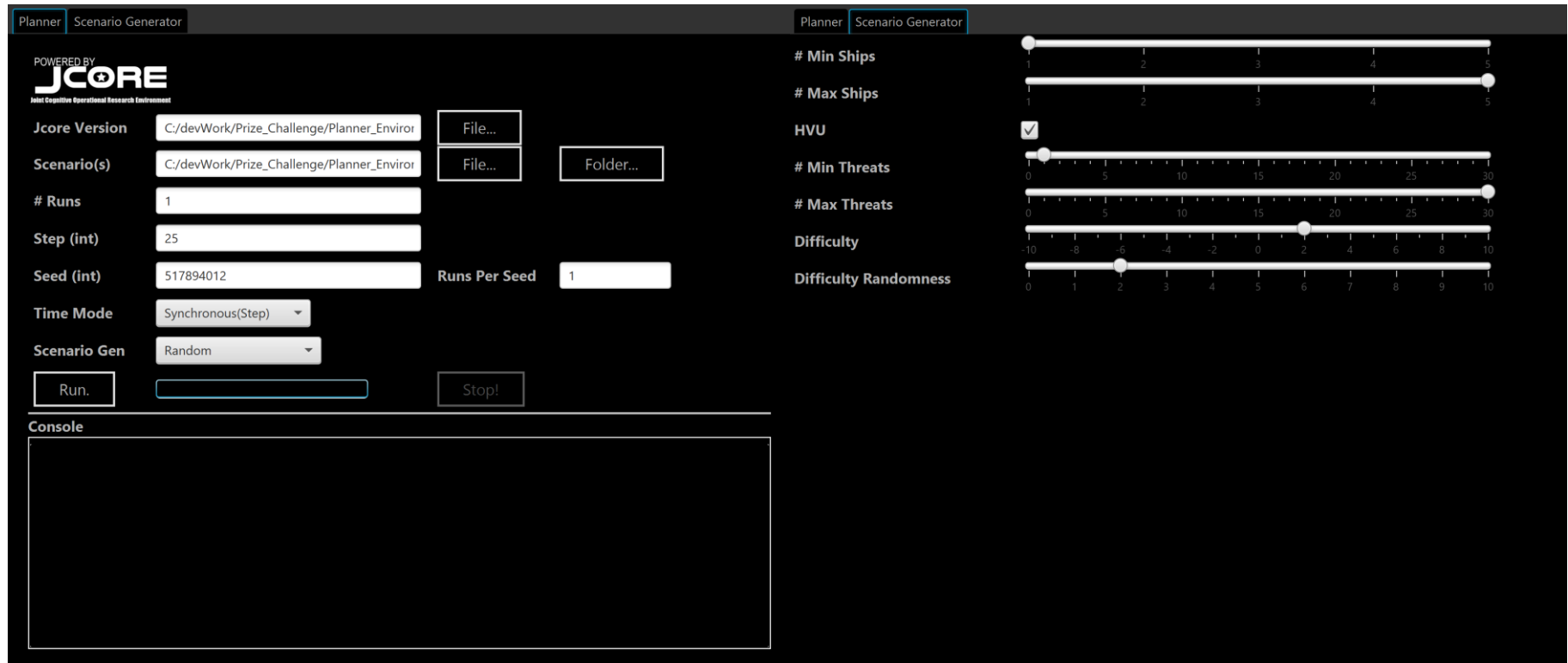
```
@SubscribeTo(StatePb.class)
public void receiveStatePb(StatePb msg) {
```


- The Python starter code is structured in a very similar way
- However, methods with “receive” in their name will be added to a list of subscribers that can receive messages
- Messages with receive<Proto name> (example seen below), will be called when the client receives the corresponding message
- pip can be used to install all the required libraries

```
# Is passed StatePb from Planner
def receiveStatePb(self, msg:StatePb):
```

Example: Graphical Scenario

- As an example, below are the **Planner** settings that would result in: full scenario randomness, jCORE GUI on, and moderate difficulty
- When the settings are set as desired, select **Run**



The screenshot displays the jCORE Planner interface, which is divided into two main sections: 'Planner' and 'Scenario Generator'. The 'Planner' section on the left contains the following settings:

- Powered by jCORE**: Mini Cognitive Operational Research Environment
- Jcore Version**: C:/devWork/Prize_Challenge/Planner_Envior (with a 'File...' button)
- Scenario(s)**: C:/devWork/Prize_Challenge/Planner_Envior (with 'File...' and 'Folder...' buttons)
- # Runs**: 1
- Step (int)**: 25
- Seed (int)**: 517894012
- Runs Per Seed**: 1
- Time Mode**: Synchronous(Step) (dropdown menu)
- Scenario Gen**: Random (dropdown menu)
- Buttons**: 'Run.' and 'Stop!' buttons are located at the bottom of the settings panel.

The 'Scenario Generator' section on the right contains the following settings:

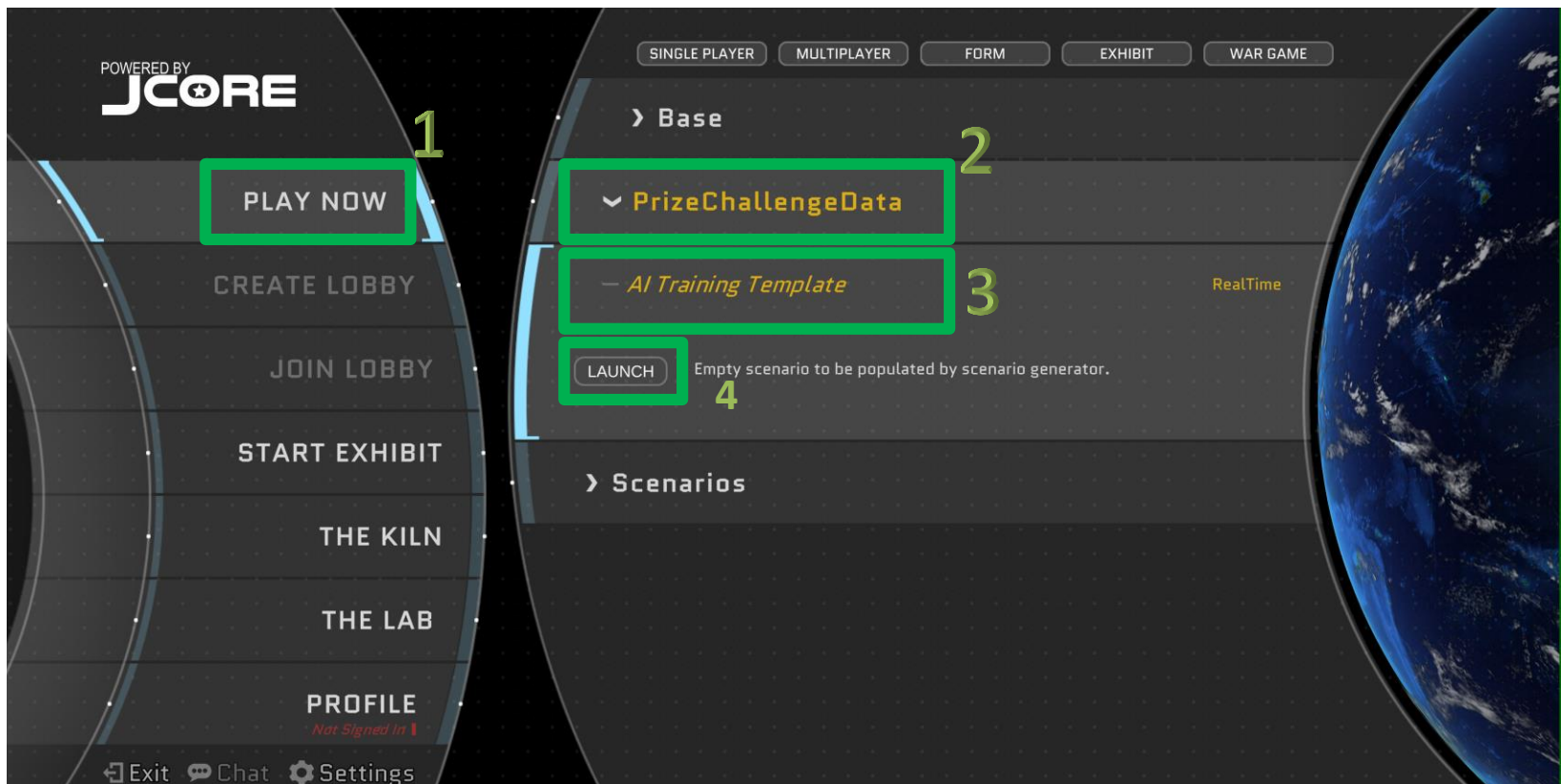
- # Min Ships**: Slider set to 1 (range 1 to 5)
- # Max Ships**: Slider set to 5 (range 1 to 5)
- HVU**: Checked checkbox
- # Min Threats**: Slider set to 0 (range 0 to 30)
- # Max Threats**: Slider set to 30 (range 0 to 30)
- Difficulty**: Slider set to 2 (range -10 to 10)
- Difficulty Randomness**: Slider set to 2 (range 0 to 10)

At the bottom of the interface is a large empty box labeled 'Console'.

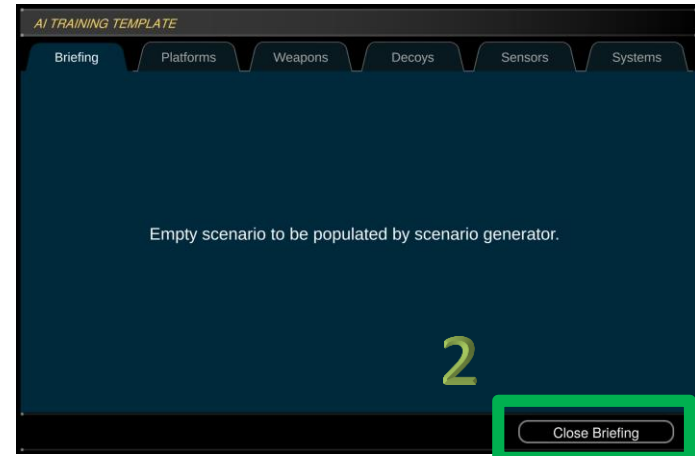
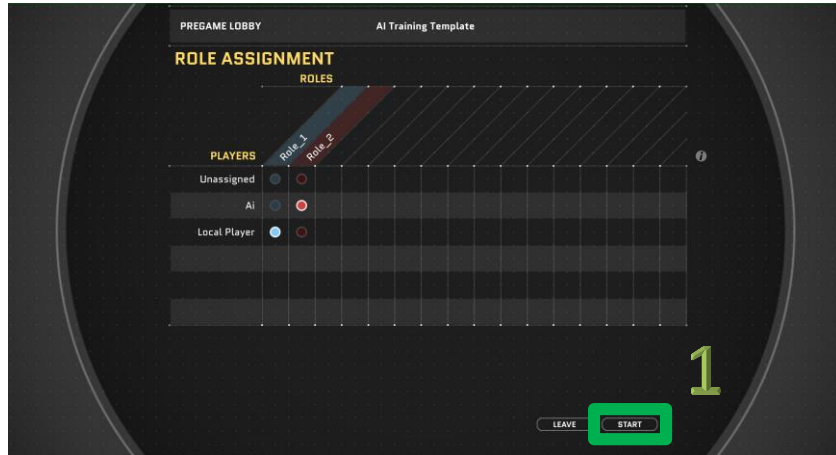
- The jCORE GUI will open and you will see the screen below
- There is no need to “login”, simply select the **x** to the right of “Welcome to jCORE”



- Select ***PLAY NOW*** > ***PrizeChallengeData*** > ***AI Training Template*** > ***Launch*** to initialize the scenario



- Select ***START*** > ***Close Briefing*** > ***OK***



- You will now be able to visually observe your AI acting

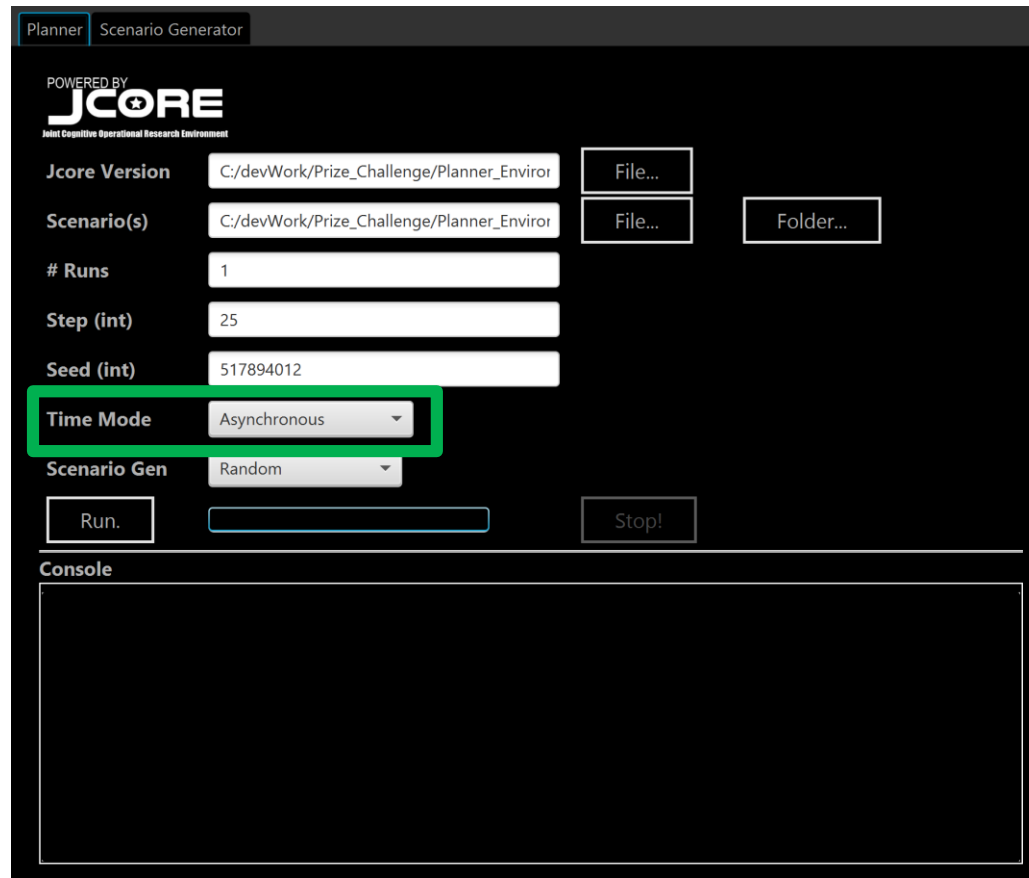


- To ensure the Planner and jCORE properly finish execution, wait until the scenario finishes, the jCORE application closes, and the Planner notifies that the run is complete
- You can alter the speed of the jCORE scenario by selected the fast forward button seen at the top of the previous slide.

WARNING: Fast forwarding increases compute resource requirements and could result in non-deterministic and undesirable behavior depending on system specifications

Example: Asynchronous

- Recall that to run in **Asynchronous** mode, the option must be selected via the **Time Mode** box seen below



The screenshot shows the JCORE Scenario Generator interface. The 'Planner' tab is selected. The interface includes the following fields and controls:

- Jcore Version:** C:/devWork/Prize_Challenge/Planner_Enviror (with a 'File...' button)
- Scenario(s):** C:/devWork/Prize_Challenge/Planner_Enviror (with 'File...' and 'Folder...' buttons)
- # Runs:** 1
- Step (int):** 25
- Seed (int):** 517894012
- Time Mode:** Asynchronous (highlighted with a green box)
- Scenario Gen:** Random
- Run/Stop controls:** A 'Run.' button, an empty text box, and a 'Stop!' button.
- Console:** A large empty text area at the bottom.

Example: Asynchronous

- As previously stated, ***Asynchronous*** mode does not require action (sending an OutputPb protobuf message) from a client to progress through the scenario
- Asynchronous runs at a non-configurable but variable speed and reports on state information every second
- The ***Scenario Generator*** tab works the same way in this mode
- When run, clients will receive State information and can still perform actions every second simulation time

WARNING

As the name implies, ***Asynchronous*** mode does not wait for the client to advance the scenario. This means that different outcomes and scores can and will be observed in ***Asynchronous*** vs ***Synchronous*** and ***GUI*** modes. In addition, client execution could be interrupted by successive state messages since the **Planner** is not waiting for the clients instruction to advance the scenario. ***Asynchronous*** mode should be used primarily in an environment testing capacity and not for AI training.

Example: Synchronous

- Similar to the other modes, to run in ***Synchronous Mode***, the option must be selected
- Unlike the other modes, setting ***Step*** will control the size of the steps in the simulation



The screenshot shows the JCORE Planner Scenario Generator interface. The 'Planner' tab is selected. The interface includes fields for 'Jcore Version', 'Scenario(s)', '# Runs', 'Step (int)', 'Seed (int)', 'Time Mode', and 'Scenario Gen'. The 'Step (int)' field is set to 25 and is highlighted with a green box. The 'Time Mode' dropdown is set to 'Synchronous(Step)' and is also highlighted with a green box. The 'Run...' button is visible at the bottom left, and the 'Stop!' button is at the bottom right. A 'Console' section is at the bottom of the window.

Planner Scenario Generator

POWERED BY
JCORE
Joint Cognitive Operational Research Environment

Jcore Version C:/devWork/Prize_Challenge/Planner_Enviror File...

Scenario(s) C:/devWork/Prize_Challenge/Planner_Enviror File... Folder...

Runs 1

Step (int) 25

Seed (int) 517894012

Time Mode Synchronous(Step)

Scenario Gen Random

Run. Stop!

Console

- ***Synchronous*** mode requires an active/running client to operate
- Before hitting ***Run***, ensure your client is open and finished initializing
- If running the example clients, you will know the client, Planner, and jCORE are working as state information will begin to print in the client console
- If clients are not sending an OutputPb after receiving a StatePb message, it will be clear as the simulation will not continue running and your client will not get additional messages

Tips for Getting Started



1. Get the Planner running with the provided client of your choice before writing your own client or making code changes to the provided ones
2. Study the provided clients close to truly understand how they interface with the Planner
3. Make small changes and custom print statements to the clients to better understand the interface and syntax for working with protobuf messages
4. Baby Steps! Before you develop your super AI, make sure you can launch missiles and that the actions are being represented in the simulation and state data

5. Start with a “CONST” scenario so the state data and actions are easy to understand and verify
6. Mess around with the ***Scenario Generator*** tab and analyze how it impacts the simulation state!
7. If your client isn’t acting as expected, log the state data and actions you take and analyze! In addition, look at the generated ***Planner*** logs in the “log” folder for possible errors
8. If you’re having serious trouble with the environment and can’t find a solution, don’t be afraid to ask for help by the information on the “Contact Information” page!

- Clicking run at the near instant the Planner GUI opens can cause the simulation to not begin depending on local system performance.
- Ensure all cmd prompt windows opened by the Planner are closed if you wish to re-run the entire Planner. Failing to do so will cause the simulation to not work.
- VPN's can cause the local IP connections between the Client, Planner, and jCORE to fail.
- If using OneDrive and things are not working, ensure that OneDrive does not have access to the Planner Environment in any way.

- Certain firewalls and antivirus applications have the potential to prevent proper operation of the Planner environment. It is recommended, but not required, to download the environment onto computers with admin privileges or local IT support to resolve these issues.
- Asynchronous, Synchronous, and GUI modes of operations are DIFFERENT. Because they change how the simulation is run and the manner of communication, they could have different results for identical scenarios and settings.
- It is recommended to use Synchronous mode when developing and evaluating AIs

- If serious issues or bugs are encountered, do not hesitate to post questions and concerns to [Challenge.Gov](https://www.challenge.gov/?challenge=artificial-intelligence-%28ai%29-and-machine-learning-%28ml%29-algorithm-development-challenge&tab=contact)
(<https://www.challenge.gov/?challenge=artificial-intelligence-%28ai%29-and-machine-learning-%28ml%29-algorithm-development-challenge&tab=contact>) via the “Contact” tab