

Project Everglades

**AI Developer’s Guide**

v1.4 | August, 2019

**Cubic | Intific**

10814 Jollyville Rd.

Building 4, Suite 350

Austin, TX 78759-5630

CONTACT INFO

**Dan Magaha**

Director of Product Development

+1 512-776—8246

dan.magaha@cubic.com

Everglades Phase 2 – SOW A – Item 2.2 a – Implementation Plan

Everglades AI Developer’s Guide

Contents

[OVERVIEW AND GAMERULES 2](#_Toc17293113)

[RUNNING A MATCH 4](#_Toc17293114)

[LOADING AND VIEWING TELEMETRY 4](#_Toc17293115)

[MOVEMENT 5](#_Toc17293116)

[COMBAT 5](#_Toc17293117)

[NODE CONTROL 5](#_Toc17293118)

[DESTROYED UNITS 5](#_Toc17293119)

[UNIT TYPES 6](#_Toc17293120)

[TRAINING A REINFORCEMENT LEARNING AGENT 7](#_Toc17293121)

[GYM ENVIRONMENTS 10](#_Toc17293122)

[OBSERVATION SPACE 10](#_Toc17293123)

[ACTION SPACE 11](#_Toc17293124)

[MAP LAYOUT 11](#_Toc17293125)

# OVERVIEW AND GAMERULES

Everglades is a synchronous, turn-based 1v1 competitive strategy game designed to be “played” by AI players. The primary objective is to capture the opponent’s base, and the secondary objectives are capturing nodes and eliminating enemy opponents.

AI players move their units, represented in the game as one of three drone classes, around the map and attempt to capture territories, destroy opponents, and learn more about the environment and enemy positions.

The game map is made up of a series of connected nodes. Each node has a specified set of connections to other nodes. Players can move units from one node to another so long as there is a valid connection between the two. The time to capture a node, the distance between nodes, and the arrangement of nodes and connections are all configurable.

A player’s team is composed of 100 units organized into 12 groups. The player directs groups to move between nodes with the purpose of capturing the opponent’s base. Units cannot be separated from their group or controlled individually.

The player is allowed to move up to 7 groups each turn; at least 5 groups must remain idle each turn. The server waits for both players to make their moves. Once the moves are received the game reconciles any combat and advances to the next turn.

There are three classes of units: Controller, Tank, and Striker. Each class has a unique advantage.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Movement | Armor | Damage | Capture Speed |
| STRIKER | **+50%** | **-50%** | **+50%** | *base* |
| CONTROLLER | *base* | *base* | *base* | **+50%** |
| TANK | *base* | **+50%** | *base* | *base* |

Figure 1: unit class attributes

The Controller captures nodes faster, making it well-suited for capturing uncontested nodes. The Tank class has extra armor, making it a better choice for defending. The Striker has increased speed and does more damage, making it more effective at attacking enemy units, but it is also significantly lighter and susceptible to damage as a result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group | Loadout |  | Group | Loadout |
| Group A | 8 Strikers |  | Group G | 8 Strikers |
| Group B | 8 Controllers |  | Group H | 8 Controllers |
| Group C | 8 Tanks |  | Group I | 8 Tanks |
| Group D | 8 Strikers |  | Group J | 8 Strikers |
| Group E | 8 Controllers |  | Group K | 8 Controllers |
| Group F | 8 Tanks |  | Group L | 12 Tanks |

Figure 2: Starting unit group loadout

The game ends when one team captures the other team’s base node, the turn limit (150) is reached, or if both teams are completely eliminated. The game does not end if one team’s units are completely eliminated, the remaining team must still capture the opponent’s base to end the game.

# RUNNING A MATCH

When you boot up the client app, you can choose to **connect to a server** or **load an existing telemetry file**. When connecting to a server, the client will launch a server in the background.

Next, you will select the two AI scripts that will participate in the match, and then click Start.

When the server starts, both AI’s will connect and send a “Ready” message. Once both AI’s are ready, the match will begin and be resolved. The client will immediately go into telemetry playback mode.

## LOADING AND VIEWING TELEMETRY

Loading existing telemetry allows you to run previous matches for playback without running a server. You will simply select an existing match telemetry file, and the match level will load. Once loading is complete, the match will be played in a cinematic “Esports” presentation mode.

You have free-roaming camera controls in this mode. The “WASD” keys controls lateral camera movement, while “Q” and “E” keys move up and down, respectively.

At the bottom of the screen is the match timeline and the “Play” button at the bottom. Clicking “Play” will begin playback (and can be toggled as a pause button once playback has started). The timeline can be scrubbed back and forth. To the right of the timeline, there is an Exit Mission button that will simply close the application at any time.

On the left side of the screen, there is a list of team groups. Initially, each team will have one group, but as orders are executed, more groups will appear. Double clicking on a group will focus the camera on that group’s movement as a Chase Camera.

Each Node is represented in a section below the groups that show the capture state of the Nodes. Double-clicking a node jumps to a fixed camera view of that node. At any time, pressing a “WASD” key, “Q” or “E” will return to Free Cam mode.

In the top center of the screen, zone control and unit status for each side is shown. There is also a timer that counts the time since the beginning of the match.

# MOVEMENT

Units move between nodes via connections. Units can transit any connection, they do not need to have captured a node to access its connections. Every connection takes time to traverse based on its distance. Combat will not occur when moving between node connections. If units from opposing teams enter from opposite sides at the same time they will simply pass each other.

# COMBAT

Combat only occurs when opposing units occupy the same node.

Unit types have different amounts of health and/or attack power. For example, Tank units have increased health while Strikers have increased attack power.

If combat occurs on a controlled node, the team controlling that node gains a defense modifier. A node can provide 1x, 1.25x, or 1.5x protection. This defense modifier will affect the direct damage applied by the opposing team. Base nodes have 1x protection, a node connected to either base has 1.5x protection, and all other nodes have 1.25x protection. Base nodes have 500 capture points while all other nodes have 100.

For example, if a Controller fires on a Striker in a node with 1.5x protection controlled by the Striker, the Controller will deal half its normal damage.

## NODE CONTROL

When a unit or group occupies an uncontrolled node it will start to claim that node for its team. The time it takes to capture the node is defined by the node’s control points and the highest node capture speed in the group that is capturing. In this edition of the game, all nodes have the same number of control points with the exception of the base nodes which have 5x as many. If at any time the other team arrives at the node, the node is considered contested and the capture timer will restart.

If a team arrives at a node controlled by the other team and no opposing units are present, they will begin to retake the node. The node must return to neutral to be retaken. Without unit bonuses, it takes 3 turns to return a node to neutral. If the node becomes contested in that time, the timer will reset. Once the node is neutral, the timer starts for claiming the node, and follows the rules above if contested.

## DESTROYED UNITS

A damaged unit does not regenerate lost health and a destroyed unit does not respawn. Once all units of a group are destroyed the group can no longer be controlled.

# UNIT TYPES

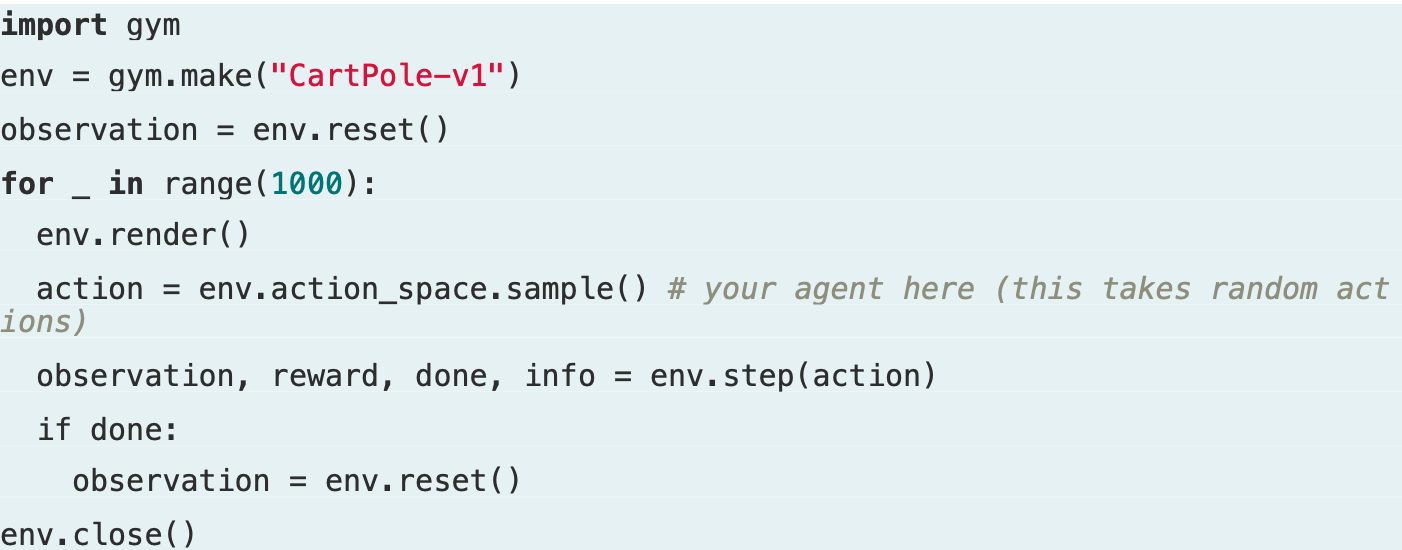
There are currently three different types of units: Controller, Tank, and Striker.

* Controller
  + Health: 100
  + Attack power: 10
  + Speed: 1
  + Cost: 1
  + Node capture speed: s/turn
* Tank
  + Health: 150
  + Attack power: 10
  + Speed: 1
  + Cost: 1
  + Node capture speed: 1/turn
* Striker
  + Health: 50
  + Attack power: 20
  + Speed: 2
  + Cost: 1
  + Node capture speed: 1/turn

# TRAINING A REINFORCEMENT LEARNING AGENT

AI agents interact with the game server via an Open AI Gym interface. The gym will provide an observation for each turn. The agent is expected to ingest the observation and send an action to the gym. This process repeats until the game ends. The agent can play as many games as needed.

The Open AI Gym interface is standard for writing reinforcement learning agents. The interface makes it easy to reuse the same interaction protocol across agents and environments. An example of the typical interaction between Gym, agent, and environment is shown below:



The above is an example of a basic training script. When training an agent the `env.action\_space.sample()` line would be replaced with an action selected by the agent. This format can be seen in the provided `agent\_template.py` file.

The only API calls required for training an agent are implemented in the above code block. A full description of all API calls supported by the Open AI Gym interface can be found [here](https://gym.openai.com/docs/).

The `agent\_template.py` implements an agent that takes random actions, it does not learn from sampled observations. It is meant to server as a boilerplate for creating more advanced agents or serving as an opponent for a reinforcement learning to play against.

The following docker images are required:

* Game server
* Gym environment
* Agent

Please note that the Gym environment image is used to create the agent image. When training/evaluating an agent there will only be two Docker containers running: the game server and agent. For almost all use cases the server can be run alongside the agent. Running these containers is handled by the provided `docker\_compose.yml` file.

If circumstances require that the game server be run on a different system from the agent then the user will need to write a custom setup procedure.

Dedicated hardware to run the game server or agent containers is not necessary for this application. All reinforcement learning algorithms can successfully be used on a single, off the shelf machine without use of GPUs. A GPU can speed up the training process but is not required. Creating an agent Docker container that can use an NVIDIA GPU will be discussed later.

To run the random agent contained in `agent\_template.py`:

1. From a terminal window, navigate to the provided zip file
2. Unzip the contents:
   1. Run `tar -zxf /path/to/everglades-dist.tar.gz -C /path/to/output`
3. Navigate to the output directory
   1. Run `ls`
   2. Confirm you see the following files:
      1. agent\_template.py – an example agent script implementing random action selection
      2. demo-agent.tar – a pre-built docker image containing using agent\_template.py
      3. docker-compose.yml – starts up the game server Docker container and two agent containers to play the game all on a single machine
      4. Dockerfile – the template used to build agent Docker images
      5. Dockerfile.gpu – the template used to build agent Docker images capable of using the machine’s GPU
      6. everglades-env.tar – a pre-built Docker image containing the everglades Gym environment
      7. everglades-server.tar – a pre-built Docker image containing the everglades game server
      8. README.txt – developer’s notes to set up the game with agents
4. Load the server image:
   1. Run `docker load -i <path/to/everglades-server.tar`
5. Load the Gym environment:
   1. Run `docker load -i <path/to/everglades-env.tar`
6. Build the agent image:
   1. Run `docker build -t everglades-agent .`
   2. Confirm you seen the “Successfully built …” and “Successfully tagged everglades-agent:latest” output
7. Confirm the images were loaded by Docker:
   1. Run `docker images`
   2. Confirm you see everglades-agent and everglades-server in the output list
8. Start up the server and agent containers
   1. Run `docker-compose up -d`
9. Confirm three containers are running:
   1. Run `docker ps`
   2. Verify the following appear under the names column:
      1. <directory\_name>\_agent\_1`
      2. <directory\_name>\_opp-agent\_1
      3. <directory\_name>\_server\_1
10. The game is currently being played by two random agents (agent\_1 and opp-agent\_1)
11. View the logs from a container:
    1. Run `docker logs -f <directory\_name>\_agent\_1`

If the reinforcement algorithm being used requires the use of GPUs, follow the below instructions to create a new agent image. Please note the server receives no benefit from using GPUs. Only a reinforcement learning agent script specifically designed to take advantage of GPUs.

1. Complete steps 1-5 from above if not already completed
2. Write an agent script optimized to use GPUs (assumed to be agent\_gpu.py)
3. Update Dockerfile.gpu:
   1. Open Dockerfile.gpu
   2. Update the line (#47): `COPY ray\_agent.py ./main.py` to `COPY agent\_gpu.py ./main.py`
4. Build a new agent Docker image:
   1. NOTE: This must be done from a Linux machine with CUDA
   2. Run `docker build -t everglades-agent -f ./Dockerfile.gpu .`
5. Complete steps 7-11 from above

Additional notes on using GPUS:

* A Linux machine must be used to build an image using the Dockerfile.gpu
* An image built using the Dockerfile.gpu file must be run on Linux hardware. It will not work in a Linux VM as the VM does not have access to the GPU
* Use of GPUs is likely overkill in most circumstances
* The provided docker-compose.yml creates two agent containers and one game server container, it does not support a multi-machine setup

Since both agent containers and the game server container run on the same machine and Docker creates a virtual network for the containers to talk to one another, no firewall changes are needed. The ports used by the server image cannot be changed.

# GYM ENVIRONMENTS

Four environments are provided in the everglades-env image:

* Base (Everglades-v0): the base game as described in this document. The observation returned by the environment is a 105x1 matrix.
* Base with stochasticity (EvergladesStochastic-v0): this is an extension of the base environment except combat is non-deterministic.
* Image (EvergladesVision-v0): this environment is the same as the base game except the state returned is an image (NxNx3). The image size can be set using the config variables ‘frame\_width’ and ‘frame\_height’.
* Image with stochasticity (EvergladesVisionStochastic-v0): this is the same as the Image environment but with stochasticity in combat calculations

Each environment takes a config (Python dictionary) to adjust various parameters. This config can be passed to the environment using the following format:

`gym.make(env\_name, env\_config=env\_config)`

Currently the only config parameters that need to be set are the environment name (ENV\_NAME) and image size (frame\_width and frame\_height), all others can be left as defaults or as detailed in the provided docker-compose.yml.

## OBSERVATION SPACE

Below is defines each value returned by Everglades-v0 and EvergladesStochastic-v0 as observations:

**Observation index 0**:

Turn number [0-151]

**Observation indexes 1-4, 5-8, 9-12, 13-16, 17-20, 21-24, 25-28, 29-32, 33-36, 37-40, 41-44:**

Node {1-11} states

is fortress [0,1]

is watchtower [0,1]

percent controlled [-100-100], if negative, opp owned; if positive, self-owned

number of opponent units [0-100]

**Observation indexes 45-49, 50-54, 55-59, 60-64, 65-69, 70-74, 75-79, 80-84, 85-89, 90-94, 95-99, 100-104**:

Group {1-12} states

location as node number [1-11]

class number [0-2] (0: Controller, 1: Striker, 2: Tank)

average group health [0-100]

is in transit [0,1]

number of units alive [0-100]

## ACTION SPACE

All environments ingest the agent’s action through the `env.step(action)` call. The environment expects a 7x2 array. The agent is allowed to move 7 of the 12 groups each turn. Each row in the action array specifies the group number and node ID to move to.

For example, a possible action for a turn would look like:

“[[1, 3], [2, 3], [3, 3], [4, 5], [5, 5], [7, 5], [10, 5]]”

This action tells groups 1, 2, and 3 to move to node 3 and groups 4, 5, 7, and 10 to move to node 5.

The game will ignore any illegal moves and does *not* prompt the agent to make a valid move. The agent loses any invalid moves. Illegal moves occur when the node ID specified is not directly connected to the node currently occupied by the specified group, when an invalid node ID or group is specified, or when the group is in transit between nodes.

# MAP LAYOUT

The Map for the initial build will be static and can be represented by JSON for communication across the server and client. Example JSON below.

Nodes are all circular for this demo, defined by a simple radius. Connections are placed around the outside of the node, defined by a 0-1 value that defines where along the circumference the exit is located.

Maps follow the format

* Map
  + Name
  + List of MapNodes
* MapNode
  + ID
  + Radius
    - The radius of the node. All nodes are circular in the prototype.
  + ResourceType
    - The resource on this node. For the prototype, there will be no resources
  + StructureDefense
    - The defense applied to the team that controls this node. This will affect weapon damage from the opposing team.
  + TeamStart
    - Which team ID starts on this node? -1 is default
  + NodeDefense
    - How much defense does this node have? Defense will affect acquisition speed.
  + List of MapConnections
* MapConnection
  + The MapNode ID of the connected node
  + Distance

Example Map JSON:

*{*

*"\_\_type":"Map:#Everglades\_MapJSONDef",*

*"MapName":"Default",*

*"nodes":[*

*{*

*"Connections":[*

*{*

*"ConnectedID":1,*

*"Distance":5*

*},*

*{*

*"ConnectedID":2,*

*"Distance":5*

*},*

*{*

*"ConnectedID":3,*

*"Distance":5*

*}*

*],*

*"ID":0,*

*"Radius":10,*

*"Resource":0,*

*"StructureDefense":1*

*"NodeDefense":5*

*"TeamStart":0*

*}*

*]*

*}*