

# Wargus Challenge Problem

Matthew Molineaux<sup>1,2</sup> and David W. Aha<sup>2</sup>

<sup>1</sup>ITT Industries

Advanced Engineering & Sciences Division  
Alexandria, VA 22303

<sup>2</sup>Intelligent Decision Aids Group

Navy Center for Applied Research in Artificial Intelligence  
Naval Research Laboratory (Code 5515)  
Washington, DC 20375  
{aha,molineau}@aic.nrl.navy.mil

## 1. Overview

The *Stratagus* Real-Time Strategy Gaming engine is an open-source project created for the development of real-time strategy games. The project's website, at <http://www.stratagus.org>, contains information about downloading and installing the engine, as well as obtaining the most popular dataset, which is from the commercial game *Warcraft II* made by Blizzard. The game played when using this dataset with *Stratagus* is called *Wargus*.

The Testbed for the Integration and Evaluation of Learning Techniques (TIELT) is a free software system developed at the Naval Research Laboratory (NRL) to permit the use of complex, large state-space games as a testbed for Machine Learning. Among other uses, it allows a researcher to connect a software system, called a *decision system* in TIELT for its role in making decisions that control the outcome of the game, to game engines with instrumented APIs. For more information about TIELT, including a manual, tutorial, and slides, please visit the TIELT website at <http://nrlsat.ittid.com>.

Lehigh University, in collaboration with NRL, has developed an API for communications between TIELT and Wargus. This integration allows a TIELT-connected decision system to access certain features of the game state and perform certain actions within the game. This integration does not yet encompass the full range of sensory information and command addressability available to a human player, but it does allow the decision system to make choices that control the outcome of the game.

## 2. Challenge Problem and Challenger Responsibilities

The challenge problem posed here concerns transferring learned strategies for winning the game of Wargus, where the positions of the player and opponent during the training phase are exchanged during the testing phase. For the purposes of this challenge problem, a map will be provided that defines starting locations for two players, one of which will be a decision system designed by the researcher, and the other will be a static opponent. This map will be used for each run of the game. One of these two positions will be consistently assigned to the decision system during a training period; however, the other position will be used for testing the system's performance. Information about choices

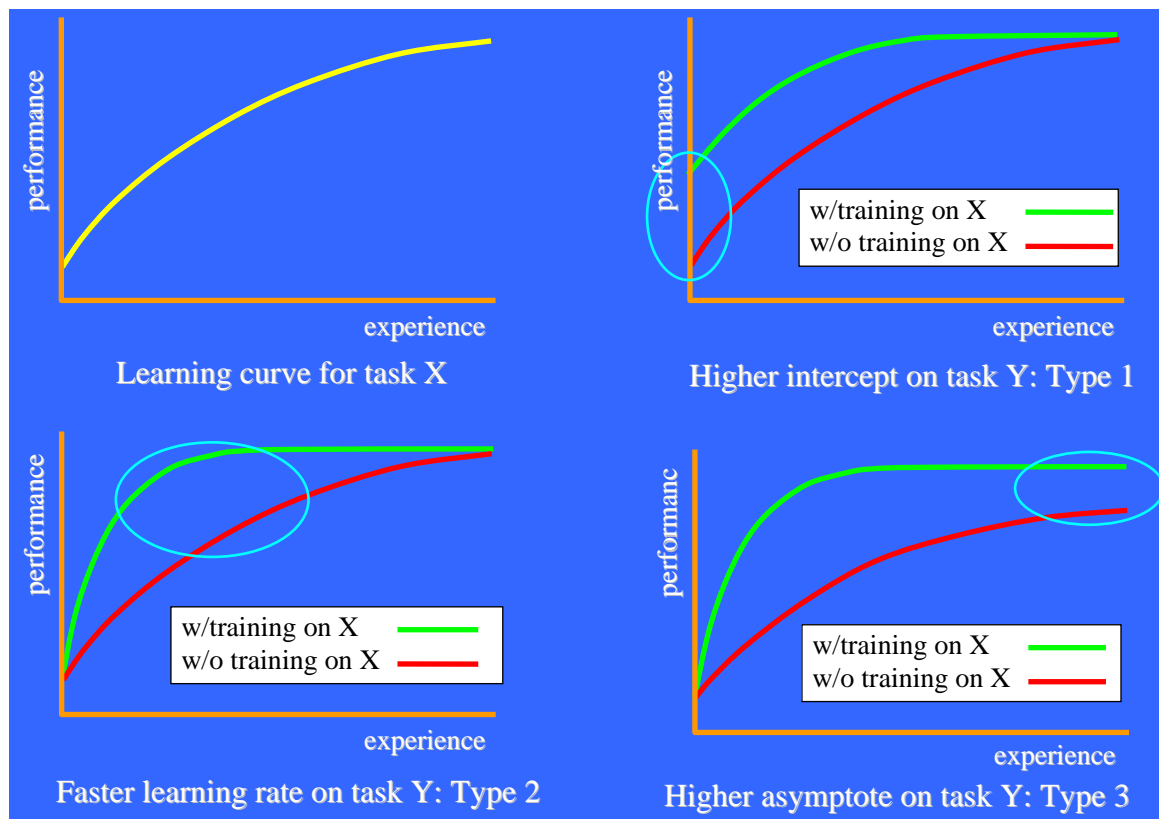
made by the opponent will be available to the decision system, which can be used for forming a model of the current state, as well as for modeling potential strategies to be used during the testing phase.

Using TIELT, NRL will create and provide an *Experiment Methodology*, a *Game Model*, a *Game Interface Model*, and a partial *Agent Description* for use with the game Wargus. These are four of the five knowledge bases that control TIELT integrations. Researchers attempting the challenge problem must create their own *Decision System Interface Model* and provide a decision system to perform the task through TIELT.

The Experiment Methodology will describe an experiment, to be executed 5 times (to acquire average results), in which the game is run  $m = \{0, 25, 50, 75, 100\}$  times during the training phase with the decision system controlling one set of forces, followed by a testing phase, which involves running  $n = 100$  games with the decision system controlling the second set of forces. The decision system will be notified to wipe its training data after the end of each testing phase.

### 3. Evaluation

The goal of this challenge problem is to transfer knowledge gained to the new task. The decision system will be evaluated on three types of improvement: higher intercept (type 1), faster learning rate (type 2), and higher asymptote (type 3) for greater values of  $m$ .



Performance on the task will be measured as a combination of the game score and the victory condition. Given  $W = \begin{cases} 1, & \text{if } \_Wins \\ 0, & \text{if } \_Loses \end{cases}$ ,  $S$  is the integer value for the final score attained in a match, and  $O$  is the integer value of the opponent's final score, the performance  $P = \frac{1}{2} \left( W + \frac{S}{(O+S)} \right)$ .

#### 4. Wargus Game Model

TIELT's Game Model defines the accessible state of a game, the operators that a system can use to affect the game, and the events that can occur during the game. Researchers attempting this problem will have to become familiar with this, because it represents the interface to the game. The following should be used as a reference to important aspects of the Wargus Game Model.

For more information on TIELT's Game Model and related topics, please see our on-line User's Manual or Tutorial (available at <http://nrlsat.ittid.com>).

##### 4.1 Types

Two object types are defined within the Game Model that will be important to systems using the Wargus integration.

*ForceParam* – This defines a set of units of one type. It consists of the *unit type name* and an integer value indicating a number of units to be used.

*Force* – This is an abstraction for an army of units. All units within a force can be directed together. It consists of an array of *ForceParam* objects and an integer *id*. This id should be a digit between 0 and 9; some of these numbers may have special meaning to the engine. Once an id is duplicated, the earlier unit with that id can no longer be given orders. It also contains a Boolean value *ready* which identifies whether the force has been completed, and a String *activity* which identifies the army's state, whether it be attack, defense, or neither.

##### 4.2 Operators

Operators are used within the agent to affect the state of the game.

*Build(building : String)* – This operator is used to tell your workers to start creating a building. The single argument must be a legal building name. This operator will have no effect if the building is not buildable at your current stage of technology.

*BuildForce(army : Force)* – This operator orders the building of a new army.

*DefendWithForce(army : Force)* – This operator orders a completed army into a defensive stance.

*Fight(army : Force)* – This operator orders a completed army to attack the enemy. No control is given over where to attack.

*GetCycle()* – Queries the game engine for the cycle time, which is the number of ticks since the game was started.

*Research(advancement : String)* – Orders a new advancement to be researched. This must be a legal advancement string. If the advance cannot be researched in the current building state, this will have no effect.

*SetMaintain(workers : Integer)* – This sets a minimum number of a certain unit type to maintain at all times. Normally used to make certain that adequate workers exist for building and gathering tasks.

*QuitGame()* – Quits the game.

*GetStat(player : Integer, statType: String)* – for the purposes of this challenge problem, the numeral 0 will always identifier the player controlled by TIELT, and the numeral 1 identifies the opponent. The value for statType must be one of the legal statistic request strings.

*Upgrade(building : String)* – Upgrades one building to another. For example, upgrading the City Hall to a Keep would require the operator ‘Upgrade(“Keep”)’.

### 4.3 Events

*BuildingBuilt* – occurs when the building that the system has registered to wait for is completed.

*NewBuildingInfo* – occurs when a building is built, by either side. Information on the name of the building and which side built it is available.

*ForceCompleted* – occurs when the force that the system has registered to wait for is completed.

*NewForceInfo* – occurs when a new unit is built, by either side. The type of the unit, and which player built it, will be made known.

*TimeSensed* – occurs when the time (as a number of cycles) is sent by Wargus (as a result of the “GetCycle()” call).

*TotalsUpdated* – occurs when a statistic is updated by Wargus (as a result of the “GetStat” call). The requested statistic will be updated in the game model.

## Appendix – Names

Unit names (to be used in creating forces)

AiWorker()  
AiSoldier()  
AiShooter()  
AiEliteShooter()  
AiCavalry()  
AiCavalryMage()  
AiMage()  
AiCatapult()  
AiScout()  
AiFlyer()  
AiPlatform()  
AiTanker()

AiSubmarine()  
AiDestroyer()  
AiBattleship()  
AiTransporter()

#### Building Names (to be used with the “Build” operator)

AiCityCenter()  
AiBetterCityCenter()  
AiBestCityCenter()  
AiLumberMill()  
AiBlacksmith()  
AiTower()  
AiGuardTower()  
AiCannonTower()  
AiHarbor()  
AiRefinery()  
AiFoundry()  
AiScientific()  
AiStables()  
AiTemple()  
AiMageTower()  
AiAirport()  
AiBarracks()

#### Advancement names (to be used with the “Research” operator)

AiUpgradeArmor1()  
AiUpgradeArmor2()  
AiUpgradeWeapon1()  
AiUpgradeWeapon2()  
AiUpgradeMissile1()  
AiUpgradeMissile2()  
AiUpgradeCatapult1()  
AiUpgradeCatapult2()  
AiUpgradeShipArmor1()  
AiUpgradeShipArmor2()  
AiUpgradeShipCannon1()  
AiUpgradeShipCannon2()  
AiUpgradeEliteShooter()  
AiUpgradeEliteShooter1()  
AiUpgradeEliteShooter2()  
AiUpgradeEliteShooter3()  
AiUpgradeCavalryMage()  
AiCavalryMageSpell1()  
AiCavalryMageSpell2()

#### Statistic Types (for use with “GetStat” operator)

Name  
RaceName  
Resources, *<resource name>*  
UnitTypesCount, *<race-specific unit or building name>*  
TotalNumUnits  
NumBuildings  
Supply  
Demand  
UnitLimit

BuildingLimit  
TotalUnitLimit  
Score  
TotalUnits  
TotalBuildings  
TotalResources, <resource name>  
TotalRazings  
TotalKills

### Resource Names

time  
gold  
wood  
oil  
ore  
stone  
coal

### Human Building Names (for use with “GetStat” operator)

unit-town-hall  
unit-keep  
unit-castle  
unit-peasant  
unit-elven-lumber-mill  
unit-human-blacksmith  
unit-inventor  
unit-stables  
unit-church  
unit-mage-tower  
unit-gryphon-aviary  
unit-human-barracks  
unit-human-watch-tower  
unit-human-guard-tower  
unit-human-cannon-tower  
unit-human-shipyard  
unit-human-refinery  
unit-human-foundry  
unit-human-oil-platform

### Orc Building Names (for use with “GetStat” operator)

unit-great-hall  
unit-stronghold  
unit-fortress  
unit-peon  
unit-troll-lumber-mill  
unit-orc-blacksmith  
unit-alchemist  
unit-ogre-mound  
unit-altar-of-storms  
unit-temple-of-the-damned  
unit-dragon-roost  
unit-orc-barracks  
unit-orc-watch-tower  
unit-orc-guard-tower  
unit-orc-cannon-tower

unit-orc-shipyard  
unit-orc-refinery  
unit-orc-foundry  
unit-orc-oil-platform

**Human Unit Names (for use with “GetStat” operator)**

unit-footman  
unit-archer  
unit-ranger  
unit-knight  
unit-paladin  
unit-mage  
unit-ballista  
unit-balloon  
unit-gryphon-rider  
unit-human-oil-tanker  
unit-human-submarine  
unit-human-destroyer  
unit-battleship  
unit-human-transport

**Orc Unit Names (for use with “GetStat” operator)**

unit-grunt  
unit-axethrower  
unit-berserker  
unit-ogre  
unit-ogre-mage  
unit-death-knight  
unit-catapult  
unit-zeppelin  
unit-dragon  
unit-orc-oil-tanker  
unit-orc-submarine  
unit-orc-destroyer  
unit-ogre-juggernaught  
unit-orc-transport