

UT-GOAL Manual

Koen Hindriks, Rien Korstanje, Wouter Pasman, Birna van Riemsdijk, and Lennard
de Rijk

Delft University of Technology, The Netherlands

April 11, 2013

1 Introduction

This manual describes the interface between GOAL and UNREAL TOURNAMENT 2004 (UT).

The manual is organized as follows. Section 2 discusses how to install the environment and how to run it. In Section 3 the initialization parameters that can be set in a `.mas2g` file are discussed. Section 4 introduces the *visualizer* that provides (i) a 3D map which displays all bots as well as the flags, (ii) options for managing the UT server, and (iii) options for managing bots. In Section 5 the UnrealGoal bot is described, a software layer between GOAL agents and UT bots that implements low-level behaviour (including e.g. a path planner). Section 6 documents the actions that the interface provides and can be performed by a GOAL agent. Finally, Section 7 documents the percepts that a GOAL agent receives when connected to the UT environment.

2 Setting up the Environment

This section explains how to startup a dedicated server to which the bots of the UnrealGoal will connect. It is assumed that you have installed GOAL and that you have installed and patched UT properly.

2.1 Installation

To install the unreal environment, you can run the installer.

2.2 Dedicated Server

There are two ways to start a server for Unreal Tournament. The first is a listen server. The server then runs inside UT and stops when UT is stopped. The second is a dedicated server. In this case the server has its own process. For the UnrealGoal environment a dedicated server is recommended.

A dedicated server can be started using the UnrealOS execution environment (UCC) from the command prompt. The ucc is located in the `Unreal Tournament 2004\System` directory.

The following command starts a game of capture the flag on the map CTF-UG-Chrome that ends after 999999 minutes.

```
ucc server CTF-UG-Chrome?game=GameBots2004.BotCTFGame?timelimit=999999
```

2.3 Web Admin

A dedicated server can be managed using the web admin. To enable it, locate `[UWeb.WebServer]` in the `UT2004.ini` and set `bEnabled=False` to `bEnabled=True`. To set a password and user name append `?AdminName=Bot?AdminPassword=Goal` to the command that starts the dedicated server. When enabled the web admin can be found at `http://localhost/ServerAdmin/`.

For your convenience a number of scripts have been provided (e.g. `CTFServer UG-Chrome`) to start the UT server and web admin. You can adopt the script to suit your own needs. For more information on managing a dedicated server please consult the UnrealAdmin wiki `http://wiki.unrealadmin.org/Server_Setup_(UT2004)`. For more command line parameters you can consult `http://wiki.unrealadmin.org/Commandline_Parameters_(UT2004)`.

2.4 Disabling Translocators

The translocator is a device that players and bots in UT can use to move across the map quickly. The UnrealGoal bot is unable to use this device. To keep things fair you can disable the translocator by locating [GameBots2004.BotCTFGame] in the UT2004.ini and adding `bAllowTranslocator=False`.

3 Connecting GOAL to Unreal Tournament

This section explains how to connect a GOAL multi-agent systems to UT by starting the UnrealGoal environment from GOAL. The UnrealGoal environment is a special .jar file that facilitates connecting GOAL to UT. In order to connect to UT, a dedicated UT server needs to be running and Pogamut Gamebots must have been installed. (See the installation instructions for this.)

The UnrealGoal environment can be launched by means of a MAS file from GOAL. The general structure and set-up of such files is discussed in more detail in the GOAL Programming Guide. Here we only discuss the initialization parameters that are specific for the UT environment.

Several parameters for initializing the UT environment can be set when connecting GOAL to UT. This is done by means of the `init` command that should be included in the `environment` section of the MAS file. For the UT environment eight parameters are available that can be set using the `init` command. All of these parameters are optional and do not have to be set using the `init` command. If a parameter is not provided, the environment will use a default value for the parameter. The following is an example of an `init` command for UT which sets all eight parameters:

```
init = [  
    botNames = ["AT1", "AT2", "AT3"],  
    visualizer = "rmi://127.0.0.1:1099",  
    botServer = "ut://127.0.0.1:3000",  
    controlServer = "ut://127.0.0.1:3001",  
    logLevel = "SEVERE",  
    skill = 3,  
    skin = "BotA",  
    team = "red"  
    startLocation = location(1,10,100)  
    startRotation = rotation(1,10,100)  
].
```

The meaning of each of the parameters is explained as follows:

- **botNames:** A list of n names that will be used by the environment to create n bots named accordingly. Names do not have to be unique. *Default value:* the empty list, meaning that no bots will be launched; bots may however also be launched by means of the Visualizer (see Section 4).
- **visualizer:** IP address of the host that is running the visualizer. If the visualizer is running on the same host as GOAL the IP address of the local host can be used.

The port should always be 1099. By connecting to the Visualizer, the Visualizer can add bots to the environment. *Default value:* none, meaning that if no address for the Visualizer is provided, the environment will not connect to the Visualizer.

- **botServer:** IP address of host running the UT server with the GameBots modifications. The bots will be created on this server. *Default value:* local host port 3000, meaning that if no address is provided the environment will connect to a server on the the local host port 3000.
- **controlServer:** IP address of host running the UT server with the GameBots modifications. This is used for the control connection to server to allow adding bots. *Default value:* local host port 3001, meaning that if no address is provided the environment will connect to a server on the the local host port 3001.
- **logLevel:** Dictates the level of detail of the log information that each bot will output to the console in the GOAL IDE. Accepted values are OFF(none), SEVERE (least), WARNING, INFO, CONFIG, FINE, FINER, FINEST and ALL (most). *Default value:* WARNING.
- **skill:** The accuracy of the bot. Any value between 1 (worst) and 7 (best). The default is 5.
- **skin:** The skin used by all bots launched from this mas. *Default value:* BotB but can be any one of:

NightMaleA, NightMaleB, NightFemaleA, NightFemaleB, MercMaleA, MercMaleB, MercMaleC, MercMaleD, MercFemaleA, MercFemaleB, MercFemaleC, EgyptMaleA, EgyptMaleB, EgyptFemaleA, EgyptFemaleB, AlienMaleA, AlienMaleB, AlienFemaleA, AlienFemaleB, BotA, BotB, BotC, BotD, Hellion_Kane, Hellion_Garrett, Hellion_Gitty, JuggMaleA, JuggMaleB, JuggFemaleA, JuggFemaleB, AbaddonM, Ophelia, JakobM, Skaarj2, Skaarj3, Skaarj4, XanM03, XanM02, XanF02, EnigmaM

- **team:** either red or blue. *Default value:* red.
- **startLocation:** The starting location of the bot provided as a location(x,y,z). When this parameter is not provided bots will spawn using the UT spawn points.
- **startRotation:** The starting rotation of the bot provided as a rotation(pitch,yaw,roll). The values of of pitch yaw and roll range from -32768 to 32767. Only the yaw value will be used by the bot. When this parameter is not provided the starting rotation is not defined.

4 Visualizer

The visualizer is a tool that can be used to visualize and manipulate the UnrealGoal environment. It can be started independently from the Environment and provides a 3D visualization of navpoints, players, bots and flags currently present in the UT server. Bots can be placed, added and removed to the UnrealGoal environment. The visualizer can also be used to place items or add them to a bots inventory.

4.1 Installation

Use the Unreal Environment installer to install the visualizer and start it via the Unreal Visualizer short cut.

4.2 Map Screen

In figure 1 you can see a simple map. The players are denoted by spheres, which are colored according to their team. Their names are also drawn above. A red arrow indicates in which direction they are looking. The blue disk and grey lines represent navigational points present in the map, these are used by automated players such as bots to move around the area. If you are playing a Capture the Flag gametype you will be able to see the flags, denoted by a sphere on a stick and once again colored according to their team.

The camera can be moved using the standard UnrealEd camera controls. To move forward/backwards hold the left mouse button and move up/down. To pan the camera hold the right mouse button down. To elevate the camera up/down hold both buttons down and move up/down, move left/right to strafe the camera. Pressing the middle mouse button will reset the camera view.

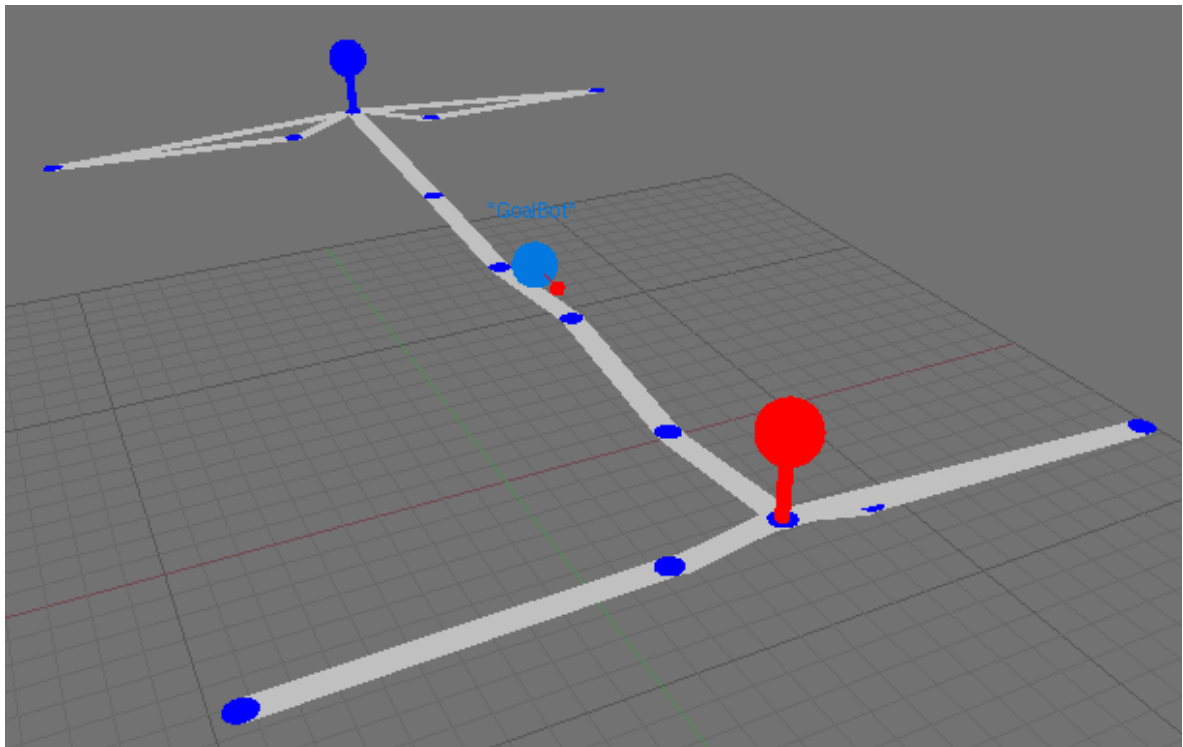


Figure 1: An example of a map screen

4.3 Server

The server menu can be found on the top of the visualizer by clicking **Server**. This section deals with the server administrative functionality.

Connection

Server->Connection will open a dialog showing the current state of the connection. When the visualizer is disconnected an address to the UT server can be provided. Pressing connect will then start the connection to the server. Should at anytime the connection to the server be lost, such as when changing a map, the Visualizer will automatically try to reconnect.

Pause / Resume

Server->Pause pauses the Unreal Server. Pausing the server means that nothing in UT will move anymore. To resume just press this same option again. Note that this will not pause individual bots that are running in GOAL.

Game Speed

Server->Game Speed allows you to change the gamespeed. A lower number means that all movement in the game will be slower, a higher number means faster movement. Although UT2004 allows for a higher gamespeed then the dialog might suggest, we have capped it to 10 to ensure that any secondary systems driving the game do not get synchronization problems on most modern pc's. Note however that it is best to leave this set to 1 to ensure normal gameplay.

Change Map

Server->Change Map will open a dialog showing a list of maps from which you can choose. Pick the one you want to switch to and press **Ok** to change the map. The server will restart and load the new map. It may take a few seconds before the visualizer notices the disconnect after which it will try to reconnect. Note that all native bots will be removed and all other bots will be disconnected.

Add Native Bot

The **Server->Add Native Bot** item will show the dialog to add an native Bot to the game as illustrated in figure 2 . All fields except level are optional. The name field allows you to set a custom name for your bot, the level field determines how accurate bot is, 1 for a total nuub and 7 for a warzone hero. Select one of the teams to assign this bot to. And finally a location where to spawn this bot, if none is selected a proper spawn location for the team the bot plays in will be used.

Add UnrealGoal Bot

The **Server->Add UnrealGoal bot** will be disabled until the Visualizer has been connected to an UT server and an environment has connected to the visualizer. As seen in figure 3 this menu is very similar to that of **Add Native Bot**. All fields except level and environment are optional. Unlike the native bots, level here only indicates how accurate the bot is. In case there are multiple environments, you can select one. When **Add Bot** is pressed, the visualizer will connect to the chosen environment and instruct it to launch another bot with the given settings.

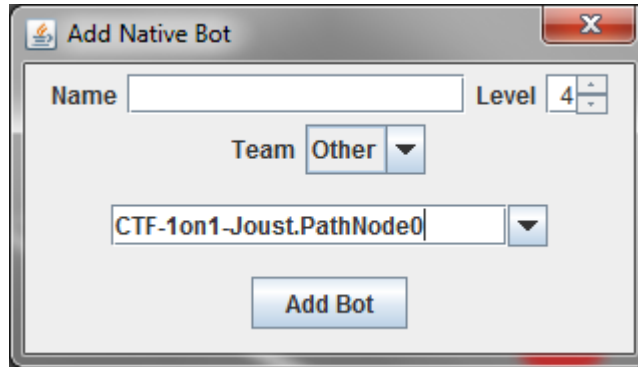


Figure 2: Dialog to add an native bot.

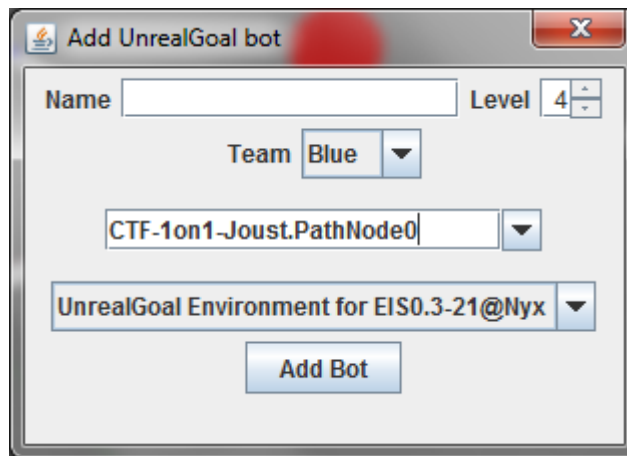


Figure 3: Dialog to add UnrealGoal bot.

List Players

The **Server->List players** item will show a list of players currently in the server. The buttons **kick** and **respawn** will respectively kick or respawn the player. Please note that actual human players can not be kicked from the server.

4.4 Context Menus

The context menu is accessible by left clicking on a player or way point. Depending on which is selected different options are available.

Respawn

When a bot is selected the respawn menu item is available (see Figure 4). Selecting it will respawn the bot at a random **PlayerStart** way point associated with their team. When a way point was selected, the menu item offers a list of bots and players to respawn on that way point. Note that native bots and players can only be spawned randomly and will only spawn at **PlayerStart** way point.

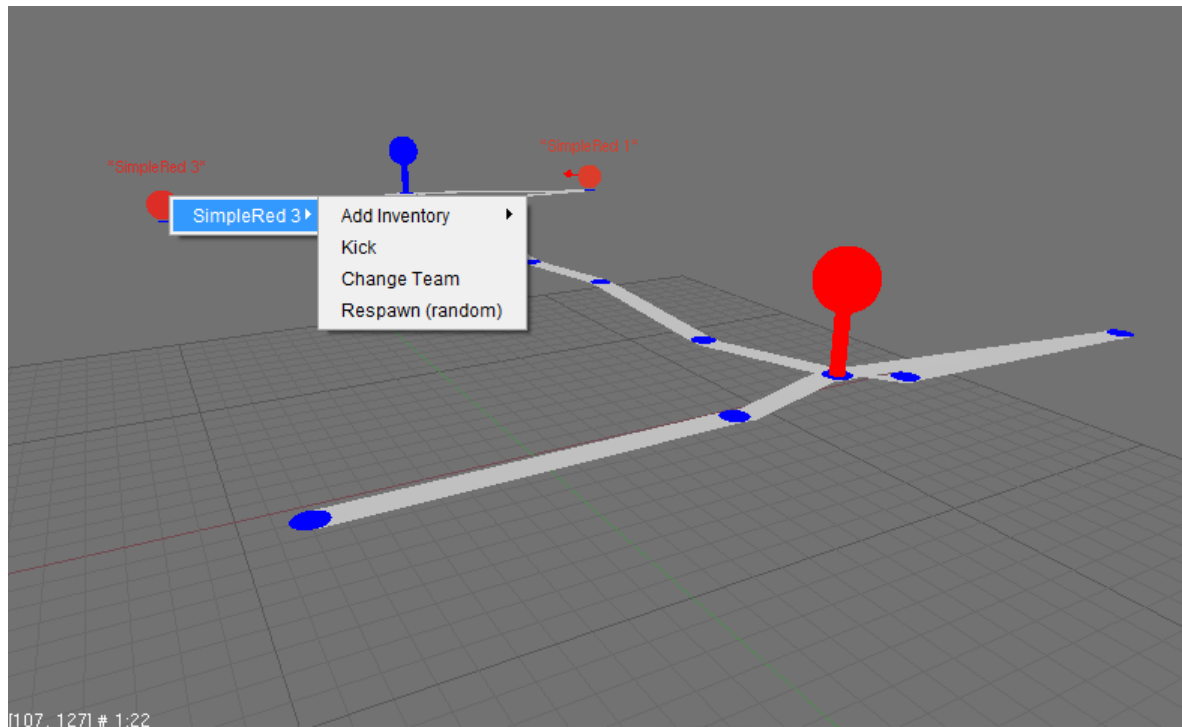


Figure 4: The context menu for a bot.

Change team of a Bot

When a bot is selected, using the change team menu item will change the team of the bot. Note that again this does not work for players and native bots.

Spawn Item

Spawn item is available whenever a way point is selected. This menu item offers a list of items that can be spawned. This will allow you to put extra weapons, health, shields or adrenaline into a level. Pickups have a respawn timer just like any other object in the world, so a newly added large shield pickup will spawn a shield every 55 seconds after it has been picked up. An example of this dialogue can be found in Figure 5.

Add Inventory

Add Inventory is available when ever a bot is selected. This menu item offers a list of items that can be added to the bots inventory such as weapons, ammunition, shields, adrenaline and health. This allows you to setup a bot without needing the bot to collect everything. Note that all weapons added in this way will not have any ammo. Ammunition has to be added in the same way as the weapon.

Kick

This menu item allows you to kick the selected bot or player from the game.

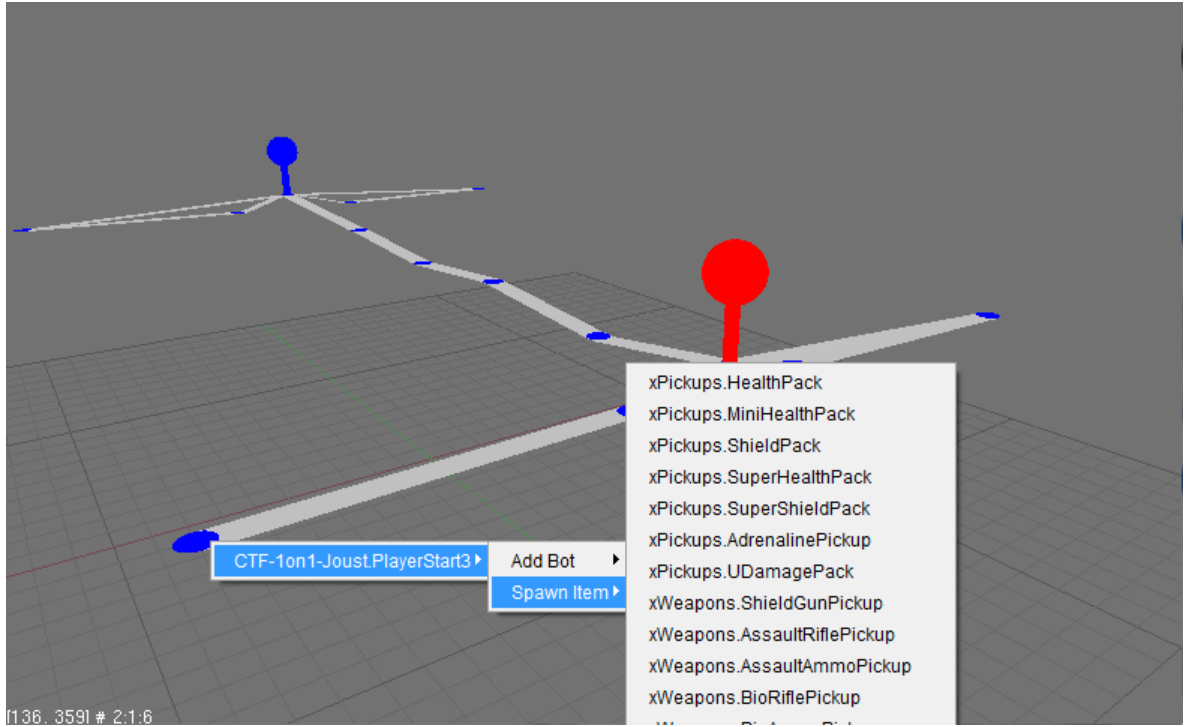


Figure 5: The context menu to spawn a new actor/item into the world.

5 UnrealGoal Bot

The UnrealGoal environment is a layer in between the GOAL agents and the UT bots that are visible in the UT environment. UnrealGoal bots implement the low-level behavior of the bots such as shooting weapons and navigating the world. This section describes how the UnrealGoal bots internally work. The aim is to create a greater understanding of the workings that can't be gained from merely reading the percepts and action specifications.

5.1 Limitations

The UnrealGoal bot has many limitations over a human player, the major limitations are:

- **Maps** Currently UnrealGoal bot is unable to do several types of movement that the native bots and human players can do.
 - Elevator Jumping. This means that the bot won't be able to time his jumps such that when he reaches the top of the elevator he gets an extra boost.
 - Precise Double Jumping. By correctly timing a second jump while mid air, bots and players can give themselves a boost and jump extra far. the UnrealGoal bot is unable to do such exact timing. This means that certain areas will be out of reach.
 - To remedy this the map needs to be modified using the Unreal Editor and all LiftExit nodes that aren't roughly at the same level as the lifts maximum have to be removed.

Translocators The translocator is a device that can teleport a player some distance and generally allow rapid movement across large maps. The bot is unable to use this weapon which provides an advantage to players and native bots that can.

- Disabling the translocator removes the advantage from other bots and players.

Currently three maps have been fixed to allow full navigation. CTF-UG-Chrome, CTF-UG-Grendelkeep and CTF-UG-AbsoluteZero. For more information about bot pathing refer to: http://wiki.beyondunreal.com/Legacy:Topics_On_Mapping.

5.2 The behavior

To abstract the low level details the Unreal Bot provides a behavior. This behavior takes input on who to shoot, where to look and where to go. This input is provided in the form of actions from GOAL. These are described below. The behavior is encoded as logic routine that evaluates approximately every 100 milliseconds. In each evaluation the following steps are taken:

- Execute all outstanding actions.
- Determine a target to shoot at based on the input from the last shoot action.
- Determine a weapon to shoot with based on the input from the prefer action.
- Determine a target to look at based on the input from the last look action.
- Determine a location to go to based on the input from the last navigate action.
- If we have a target to shoot at, shoot it with the preferred weapon.
- If we have location to go to, run to this location, look at the first applicable of the target we are shooting, the target we should look at or the path ahead.
- If we are standing still, look at the first applicable of the target we are shooting, the target we should look at or turn in circles.
- Prepare a new batch of percepts for the Environment.

Because the bot has its own thread actions are queued up and executed when the bot is ready. As a result there may be a 0-100ms delay between the execution of an action in GOAL and the bot executing the action. Likewise all percepts will be from 0-100ms in the past. This delay is small and should not adversely affect the performance of a GOAL program. However this delay should be taken into account in some cases. For example executing dropWeapon until a weapon is perceived to have been dropped may leave the bot with a big pile of weapons in front of him.

6 Actions

In this section, the design and specification of the actions that a GOAL agent can perform while controlling an Unreal bot in UT2004 is described. The actions described here are provided by an UT-GOAL interface. An overview of the actions can be found in Section 6.3.

6.1 Behavior Actions

The behavior actions are actions that govern how the bot looks, moves and shoots. The actions all take place over an extended duration of time.

Navigate action

Description	This action makes a best effort attempt to navigate the map.
Syntax	<code>navigate(<Destination>)</code>
Parameters	<code><Destination></code> : A location presented either as a location(x,y,z) or as the UnrealId of a player, navpoint, or item.
Effects	<ol style="list-style-type: none">0. Sets the destination the bot will run to.1. The bot switches to the navigation state <code>navigation(navigating, Destination)</code> ,2. Plans a path to this destination. If there is not path the navigation state becomes <code>navigation(noPath, none)</code>3. Then it starts moving along this path. If the bot gets stuck on the way the navigation state becomes <code>navigation(stuck, none)</code>4. While traveling, opponents will be shot at according to the last shoot action.5. Once the destination has been reached, the navigation state changes to <code>navigation(reached, none)</code>.
Notes	<ol style="list-style-type: none">1. A player is considered reached when the bot comes within 100 units of the player or when the player is not visible; his last known location.2. If the bot gets stuck this doesn't mean a destination is actually unreachable. It is for example possible to miss a jump twice.

Stop action

Description	Halts the bot on the spot.
Syntax	<code>stop</code>
Effects	<ol style="list-style-type: none">0. Clears the destination the bot will run to.1. The bot switches the navigation state to <code>navigation(waiting, none)</code>.2. The bot looks around as ordered by the last look action.3. If an opponent becomes visible that matches the last shoot action the bot will shoot at it.

Shoot action

Description	Sets the priority of each opponent.
Syntax	<code>shoot(<TargetLabel>)</code> <code>shoot([<TargetLabel>])</code>
Parameters	<code><TargetLabel></code> : A single target label. <code>[<TargetLabel>]</code> : A list of target labels in order of priority. A target label can be: <code>none</code> , <code>nearestEnemy</code> , <code>nearestFriendly</code> , <code>nearestFriendlyWithLinkGun</code> , <code>enemyFlagCarrier</code> , <code>friendlyFlagCarrier</code> , <code>PlayerID</code> or <code>location(X,Y,Z)</code> . The bot takes the first target label and checks if the target is visible. If the target is visible, it will be shot at. Otherwise the bot will try the next target label. When all target labels have been tried and none matched any visible players the bot will not target any opponent.
Effects	1. Sets target labels that decide which visible opponent the bot shoots at.
Notes	1. A full overview of who is targeted by which target label can be found below.

Target label	Targets
<code>none</code>	No one.
<code>nearestEnemy</code>	The nearest enemy player the bot can see.
<code>nearestFriendly</code>	The nearest friendly player the bot can see.
<code>nearestFriendlyWithLinkGun</code>	The nearest friendly player the bot can see that is holding a link gun.
<code>enemyFlagCarrier</code>	The bot from the opponent team that is currently holding the flag.
<code>friendlyFlagCarrier</code>	The bot from the team that is currently holding the flag.
<code>PlayerID</code>	The bot identified by the UnrealID. Valid Id's can be extracted from the See other bot and Flag percept.
<code>location(X,Y,Z)</code>	A location in the world.

Stop shooting action

Description	Stops the bot from shooting anyone.
Syntax	<code>stopShooting</code>
Effects	1. The bot stops shooting at once. Identical to calling <code>shoot([])</code> .

Prefer weapon action

Description	Sets the priority for the use of each weapon.
Syntax	<code>prefer(<Weapon>)</code> <code>prefer([<Weapon>])</code>
Parameters	<code><Weapon></code> : A single weapon(WeaponLabel, Firemode). <code>[<Weapon>]</code> : A list of weapon(WeaponLabel, Firemode) in the order of priority. A WeaponLabel can be: any, sniper or any of the following: <code>shield_gun</code> , <code>assault_rifle</code> , <code>bio_rifle</code> , <code>shock_rifle</code> , <code>minigun</code> , <code>link_gun</code> , <code>flak_cannon</code> , <code>rocket_launcher</code> , <code>lightning_gun</code> , <code>sniper_rifle</code> , <code>redeemer</code> . The FireMode is either primary or secondary.
Effects	<ol style="list-style-type: none">1. Set the preference used to select a weapon.2. Select the first weapon from the list that can be used (e.g. the bot has the weapon and it is loaded).3. Once the bot changes weapons it won't be able to shoot for about half a second.
Notes	<ol style="list-style-type: none">1. Using <code>prefer([])</code> will cause the bot to use its default weapon preferences.2. When the bot has none of its preferred weapons, it will use its default weapon preference.3. The default preference is: <code>weapon(shock_rifle, secondary),</code> <code>weapon(rocket_launcher, primary),</code> <code>weapon(flack_cannon, primary),</code> <code>weapon(sniper_rifle, primary),</code> <code>weapon(lightning_gun, primary),</code> <code>weapon(mini_gun, primary),</code> <code>weapon(link_gun, primary),</code> <code>weapon(bio_rifle, secondary),</code> <code>weapon(assault_rifle, primary),</code> <code>weapon(assault_rifle, secondary),</code> <code>weapon(shield_gun, secondary),</code> <code>weapon(shield_gun, primary).</code>

Look action

Description	Sets the priority of bots and locations to look at. Use this action to keep an eye on the flag or snipe down an alley.
Syntax	<code>look(<TargetLabel>)</code> <code>look([<TargetLabel>])</code>
Parameters	<code><TargetLabel></code> : A single target label. <code>[<TargetLabel>]</code> : A list of target labels in order of priority. A target label can be: <code>none</code> , <code>nearestEnemy</code> , <code>nearestFriendly</code> , <code>nearestFriendlyWithLinkGun</code> , <code>enemyFlagCarrier</code> , <code>friendlyFlagCarrier</code> , <code>PlayerID</code> or <code>location(X,Y,Z)</code> . The bot takes the first target label and checks if the target is visible. If the target is visible, it will be looked at. Otherwise the bot will try the next target label. When all target labels have been tried and none matched any visible players the bot will either look in the direction it is traveling or turn around on the spot.
Effects	1. If the bot is not actively shooting some one and the target is visible the bot will look at the given target.

Target label	Targets
<code>none</code>	No one.
<code>nearestEnemy</code>	The nearest enemy player the bot can see.
<code>nearestFriendly</code>	The nearest friendly player the bot can see.
<code>nearestFriendlyWithLinkGun</code>	The nearest friendly player the bot can see that is holding a link gun.
<code>enemyFlagCarrier</code>	The bot from the opponent team that is currently holding the flag.
<code>friendlyFlagCarrier</code>	The bot from the team that is currently holding the flag.
<code>PlayerID</code>	The bot identified by the UnrealID. Valid ID's can be extracted from the See other bot and Flag percept.
<code>location(X,Y,Z)</code>	A location in the world.

6.2 Instant actions

The instant actions are actions that can be carried out in a single instant - and with the exception of respawn - do not disrupt any ongoing behavior.

Respawn action

Description	Respawns the bot.
Syntax	respawn
Effect	1. The bot dies and is returned to a random starting position in the starting condition. 2. The navigation state is switched to navigation(waiting,none), i.e. the bot stands still.
Notes	1. The respawn action does not need to be performed if a bot is fragged. In that case, the bot is automatically respawned by UT. 2. The respawn action was originally thought up as a way out for the hopelessly stuck bot, but can also be used tactically in a CTF game. 3. The starting condition for CTF is 100 health, 0 armour, 0 adrenaline, a Shield Gun and Assault Rifle (including 100 rounds of ammo).

Combo action

Description	The bot slowly consumes 100 adrenaline pills and gets a power up.
Precondition	The bot has 100 pills.
Syntax	combo(<Combo>)
Parameters	<Combo> : Any of the following: berserk, booster, invisibility, speed. Effects: speed - Makes the bot run super fast; booster - Bot gains health at a rate of 5 points per second up to 199, then it gives the bot shield at a rate of 5 per second up to 150; invisible - Bot becomes nearly invisible. Other bots only have small chance at spotting an invisible bot; berserk - Makes you fire at a twice the rate.
Effect	1. The bot activates the power up. 2. The adrenaline slowly drains. 3. Once the adrenaline reaches 0, the effect no longer applies.
Notes	1. When invisible with the flag, the flag is still visible. This means you can still be shot at. 2. Be careful when using speed, the navigation can have trouble keeping up. 3. One adrenaline pill gives 2 adrenaline. Adrenaline stays if the bot gets killed.

Drop weapon action

Description	Drops the weapon it is currently holding, possibly for another bot to use.
Precondition	Bot is holding a weapon other than the shield gun.
Syntax	dropWeapon
Effect	1. The weapon the bot is currently holding is dropped and lands a few feet away in the direction the bot is looking.

Path action

Description	The bot computes a path from a to b.
Syntax	<code>path(<From>,<To>)</code>
Parameters	<code><From></code> : A location presented either as a location(x,y,z) or as the UnrealId of a player, navpoint, or item. <code><To></code> : A location presented either as a location(x,y,z) or as the UnrealId of a player, navpoint, or item.
Effect	1. The bot computes the path. 2. The bot emits a Path percept.

6.3 Overview

Action Name	Syntax
Navigate action	<code>navigate(<Destination>)</code>
Stop action	<code>stop</code>
Shoot action	<code>shoot(<TargetLabel>)</code> <code>shoot([<TargetLabel>])</code>
Stop shooting action	<code>stopShooting</code>
Prefer weapon action	<code>prefer(<Weapon>)</code> <code>prefer([<Weapon>])</code>
Look at action	<code>look(<TargetLabel>)</code> <code>look([<TargetLabel>])</code>
Respawn action	<code>respawn</code>
Use power up action	<code>combo(<Combo>)</code>
Drop weapon action	<code>dropWeapon</code>
Path action	<code>path(<From>,<To>)</code>

7 Percepts

This section describes the percept design that provides a specification for the implementation of the UT-GOAL interface component that handles the transfer of percepts to GOAL.

To limit the percept load, not all percepts are provided at each iteration of the cycle.

- **Send once:** “Send once” percepts are provided only once when an agent is connected for the first time to the UT environment. Handle these percepts in the **init** module of the agent.
- **On change:** “On change” percepts provide new, up-to-date information that overrides previously perceived information. These percepts are received whenever some change is perceived that invalidates old perceptual information. The new percept provides the most current and up-to-date information.
- **On change with negation:** “On change with negation” percepts negate previously observed information and, if available, provide an update. For example, if the agent perceived `flag(blue,...,location(581.0,4936.31,-742.1))` but now sees the blue flag’s position has changed it will get two percepts:
 - `percept(not(flag(blue,...,location(581.0,4936.31,-742.1))))`, and
 - `percept(flag(blue,...,location(590.37,4816.16,-742.1)))`

where the last percept provides the new, up-to-date information. These percepts thus are provided when a change occurs. In all cases of these percepts, the old percept `percept(old)` is negated and the agent perceives `percept(not(old))` which indicates the old fact is no longer true. In case a new relevant observation is available, another percept will be available that provides this information.

7.1 Self Percepts

All the following percepts relate to the bot's state in the game. Its name, physical state and inventory.

Self percept

Description	Information about the bot's identity and team.
Type	Send once
Syntax	<code>self(<UnrealID>, <NickName>, <Team>)</code>
Parameters	<p><code><UnrealID></code>: Unique identifier string for this bot assigned by Unreal.</p> <p><code><NickName></code>: Name of the bot as it appears in the game.</p> <p><code><Team></code>: Either one of the following strings red, blue, or none.</p>

Orientation percept

Description	Information about the bot's orientation in the world.
Type	Send once
Syntax	<code>orientation(<Location>, <Rotation>, <Velocity>)</code>
Parameters	<p><code><Location></code>: the position in the world as <code>location(x,y,z)</code>.</p> <p><code><Rotation></code>: the rotation of the bot as <code>rotation(pitch,yaw,roll)</code>.</p> <p><code><Velocity></code>: the velocity of the bot as <code>velocity(vx,vy,vz)</code> in ut units per second.</p>

Status Percept

Description	Information about the bot's current physical state.
Type	Send on change
Syntax	<code>status(<Health>, <Armour>, <Adrenaline>, <Combo>)</code>
Parameters	<p><code><Health></code>: An integer between 0 and 199, indicating the health left in this bot.</p> <p><code><Armour></code>: An integer between 0 and 150, indicating the amount of armour this bot has.</p> <p><code><Adrenaline></code>: An integer between 0 and 100 indicating how much adrenaline this bot has.</p> <p><code><Combo></code>: Either booster, berserk, invisibility, speed or none, indicating the active combo.</p>

Score percept

Description	Information about the number of kills, deaths, and suicides this bot accumulated.
Type	Send on change
Syntax	<code>score(<TotalKills>, <TotalDeaths>, <TotalSuicides>)</code>
Parameters	<code><TotalKills></code> : Number of people the bot fragged during this game. <code><TotalDeaths></code> : Number of times the bot died during this game. <code><TotalSuicides></code> : Number of times the bot got himself killed.
Notes	1. Respawn and being fragged by an opponent both count as a death. Being killed by the your own weapon counts as a suicide.

Current Weapon percept

Description	Information on which weapon the bot is currently holding.
Type	Send on change
Syntax	<code>currentWeapon(<WeaponType>,<FireMode>)</code>
Parameters	<code><WeaponType></code> : none or any of the following: <code>shield_gun</code> , <code>assault_rifle</code> , <code>bio_rifle</code> , <code>shock_rifle</code> , <code>minigun</code> , <code>link_gun</code> , <code>flak_cannon</code> , <code>rocket_launcher</code> , <code>lightning_gun</code> , <code>sniper_rifle</code> , <code>redeemer</code> . <code><FireMode></code> : either none, primary or secondary, indicating if the weapon is used and in which fire mode.
Notes	1. Not all weapons may be present on each map.

Inventory Weapon percept

Description	Information on which weapons the bot currently has in its inventory and their status.
Type	Send on change, with negation.
Syntax	<code>weapon(<WeaponType>, <Ammo>, <AltAmmo>)</code>
Parameters	<code><WeaponType></code> : none or any of the following: <code>shield_gun</code> , <code>assault_rifle</code> , <code>bio_rifle</code> , <code>shock_rifle</code> , <code>minigun</code> , <code>link_gun</code> , <code>flak_cannon</code> , <code>rocket_launcher</code> , <code>lightning_gun</code> , <code>sniper_rifle</code> , <code>redeemer</code> . <code><Ammo></code> : An integer between 0 and specific maximum for the <code>WeaponType</code> . <code><AltAmmo></code> : An integer between 0 and specific maximum for the <code>WeaponType</code> .
Notes	1. Not all weapons may be present on each map. 2. Depending on the <code>WeaponType</code> , <code>Ammo</code> and <code>AltAmmo</code> may refer to the same reservoir of ammunition. For more information please consult the Unreal Documentation. 3. The <code>shield_gun</code> primary has infinite ammo which is always 1. 4. The <code>shield_gun</code> secondary will recharge when not used.

7.2 Action Percepts

Fragged percept

Description	This percept is provided when one bot is violently fragmented by another.
Type	Always
Syntax	<code>fragged(<Time>,<KillerID>,<VictimID>,<WeaponType>)</code>
Parameters	<code><Time></code> : The UT time of the Frag, in milliseconds. <code><KillerID></code> : The UnrealID of killer or none. <code><VictimID></code> : The UnrealID of the victim. <code><WeaponType></code> : Weapon type used for the killing shot.
Notes	1. When the killer and victim id are equal, the bot committed suicide. 2. When the killer id is none, the bot used respawn.

Navigation state percept

Description	This percept is provided when one bot is violently fragmented by another.
Type	On change
Syntax	<code>navigation(<State>,<Destination>)</code>
Parameters	<code><State></code> : The UnrealID of killer or none. <code><Destination></code> : The UnrealID of the victim.
Notes	1. Either navigating, stuck, noPath, reached or waiting, indicating the state of the navigation. 2. For more information see the navigate action.

Path percept

Description	This percept is provided when a <code>path(<Location>,<Location>)</code> action successfully completes.
Type	On change
Syntax	<code>path(<Length>,[<Location>])</code>
Parameters	<code><Length></code> : The Length of the path in UT units. <code>[<Location>]</code> : A list of UnrealIDs that represent a path from the starting location to the end.
Notes	1. The first and last node in the path equal the <code><From></code> and <code><To></code> arguments of the Path action only iff both were navpoints. Otherwise it will be the closest navpoint.
Notes	2. Likewise the distance of the path is the distance between the starting navpoint and the destination navpoint.

Logic Iteration percept

Description	This percept indicates the number of logic evaluations of the bot.
Type	On change
Syntax	<code>logic(<Iterations>)</code>
Parameters	<code><Iterations></code> : Numbers of logic iterations since the start.

7.3 Map percepts

The following percepts provide information about what the bot knows about the map and the game. Navigations points, pick up points, etc. but also the map name, game played, current score and victory conditions.

Navpoint percept

Description	Information about point in the map. Together these form a directed graph that spans the entire map.
Type	Send once.
Syntax	<code>navPoint(<UnrealID>, <Position>, [<NeigsUnrealID>])</code>
Parameters	<p><code><UnrealID></code>: The the unique ID for this navpoint.</p> <p><code><Position></code>: The location of this navpoint on the map as a location(x,y,z).</p> <p><code>[<NeigsUnrealID>]</code>: A list of ID's for the neighbouring navpoints that are reachable from this navpoint.</p>

Item Pickup percept

Description	Information indicating at which navpoint weapons can be found.
Type	Send once.
Syntax	<code>pickup(<UnrealID>, <Label>, <ItemType>)</code>
Parameters	<p><code><UnrealID></code>: The UnrealID of the navigation point this pickup spot is placed on.</p> <p><code><Label></code>: The category of the pick up, useful for prioritizing.</p> <p><code><ItemType></code>: The actual item type of the item located on the pickup.</p>
Notes	<ol style="list-style-type: none"> 1. Depending on the game setting “weapon stay”, there may not always be a weapon present on a pick up point. 2. If “weapon stay” is enabled, a weapon can only be picked up if one of the same type is not present in the bots inventory yet. 3. A full overview of which category Label belongs to which item type.

<Label>	<ItemType>
weapon	shield_gun, assault_rifle, bio_rifle, shock_rifle, minigun, link_gun, flak_cannon, rocket_launcher, lightning_gun, sniper_rifle, redeemer
ammo	shield_gun, assault_rifle, bio_rifle, shock_rifle, minigun, link_gun, flak_cannon, rocket_launcher, lightning_gun, sniper_rifle, redeemer
health	health, mini_health, super_health
armor	small_armor, super_armor
adrenaline	adrenaline
other	udamage

Flag base percept

Description	Information about the location of the flag base. This will be the place to be for all bots.
Type	Send once.
Syntax	<code>base(<Team>, <UnrealID>)</code>
Parameters	<code><Team></code> : The team this flagbase belongs to. <code><UnrealID></code> : The UnrealID of the navpoint this flagbase is placed upon.
Notes	1. Defend well. 2. The flag may not always be present.

Game info percept

Description	Information about the type of game being played, the map and the score required for winning the game.
Type	Send once.
Syntax	<code>game(<Gametype>, <Map>, <GoalScore>, <TimeLimit>)</code>
Parameters	<code><Gametype></code> : A String representing the game. For CTF typically 'BotCTFGame'. <code><Map></code> : A string with the name of the map being played on. <code><GoalScore></code> : The first team to reach the GoalScore wins the game. For CTF this is the number of times the flag must be captured. If the goal score is zero, the game can't be won by reaching the goalscore first. If there is a time limit, the team with the highest score at the end of the game wins. <code><TimeLimit></code> : The remaining duration of the game from the moment this percept is received. When the time is up and the GoalScore is in a tie, the game will go into overtime. If the time limit is zero, the game won't end until the goal score is reached.
Note	If both the GoalScore and TimeLimit are zero, the game can't be won.

Team score percept

Description	Percept that provides information about the current state of the game.
Type	On change
Syntax	<code>teamScore(<TeamScore>, <OpponentTeamScore>)</code>
Parameters	<code><TeamScore></code> score of the team this bot is on. <code><OpponentTeamScore></code> score of the opponent team.
Notes	1. For CTF the score is the number of times the flag has been captured. 2. Once either team reaches the goal score from the Game-info percept, the game is over.

Flag status percept

Description	Percept that provides information about the current state of the flag.
Type	On change with negation.
Syntax	<code>flagState(<Team>, <FlagState>)</code> <code><Team></code> : Colour of the flag, matches the colour of the team that has to protect this flag. <code><FlagState></code> : State of the flag. Either home, held or dropped.
Notes	1. See also the See flag percept.

7.4 See Percepts

The following percepts cover what the bot can see.

See item percept

Description	Provides information about an item the bot sees in the world.
Type	On change, with negation.
Syntax	<code>item(<UnrealID>, <Label>, <ItemType>, <Position>)</code>
Parameters	<code><UnrealID></code> : The UnrealID of this item. <code><Label></code> : The category of the pick up, useful for prioritizing. <code><ItemType></code> : The actual item type of the item located on the pickup. <code><Position></code> : The x,y,z coordinates of the item as a location(x,y,z) or the UnrealID of a navpoint if the item is spawned on one.
Notes	1. A full over view of which category Label belongs to which item type.

<Label>	<ItemType>
weapon	shield_gun, assault_rifle, bio_rifle, shock_rifle, minigun, link_gun, flak_cannon, rocket_launcher, lightning_gun, sniper_rifle, redeemer
ammo	shield_gun, assault_rifle, bio_rifle, shock_rifle, minigun, link_gun, flak_cannon, rocket_launcher, lightning_gun, sniper_rifle, redeemer
health	health, mini_health, super_health
armor	small_armor, super_armor
adrenaline	adrenaline
other	udamage

See flag percept

Description	Percept provided when the flag becomes visible to the bot.
Type	On change, with negation.
Syntax	<code>flag(<Team>,<HolderUnrealID>, <Position>)</code>
Parameters	<Team> : Colour of the flag, matches the colour of the team that has to protect this flag. <HolderUnrealID> : The Unreal ID of the bot that holds flag, none when the flag is not held. <Position> : The x,y,z coordinates of the flag as a location(x,y,z).
Notes	1. See also the flagStatus percept.

See other bot percept

Description	Percept provided when another bot becomes visible to this bot.
Type	On change with negation.
Syntax	<code>bot(<UnrealID>, <Team>, <Position>, <Weapon>, <FireMode>)</code>
Parameters	<UnrealID> : Unique identifier string for this bot assigned by Unreal. <Team> : The team of this bot. <Position> : The x,y,z coordinates of the bots position in UT units, described as location(x,y,z) <Weapon> : The weapon the bot is holding. none or any of the following: <code>shield_gun</code> , <code>assault_rifle</code> , <code>bio_rifle</code> , <code>shock_rifle</code> , <code>minigun</code> , <code>link_gun</code> , <code>flak_cannon</code> , <code>rocket_launcher</code> , <code>lightning_gun</code> , <code>sniper_rifle</code> , <code>redeemer</code> . <Shooting> : Either primary, secondary or none to indicate if the bot is shooting and in which firemode.

See Navpoint percept

Description	Information about point in the map the bot can see.
Type	On change with negation.
Syntax	<code>navPoint(<UnrealID>, <Position>, [<NeigsUnrealID>])</code>
Parameters	<UnrealID> : The the unique ID for this navpoint. <Position> : The location of this navpoint on the map as a location(x,y,z). [<NeigsUnrealID>] : A list of ID's for the neighbouring navpoints that are reachable from this navpoint.
Notes	1. The See Navpoint percept is provided when ever the bot can see a navpoint that has a pickup on it. This allows you to confirm if an item has been picked up or not. If you see the navpoint, but not the item, the item has been picked up.

8 Changes

The changes since version 2 of the Unreal Goal bot.

- **Bots can now use all weapons properly.** This is done using a Pogamut Addon. Each weapon of the standard weapons have been given their own shooting that handles charging up and shooting in an intelligent fashion. Some examples:

- In Secondary mode the shield gun will selectively use its shield when threatened and bounce back incoming shock cores.
- Using the shock rifle the bot is able to pull shock combos and won't shoot a shielded player. (Aims above instead). Beware the bot will even blow up your orbs!
- In Secondary mode the BioRifle will charge up till full, then when a target becomes visible, unload the full charge. After this it will pelt its target with primary mode.
- When charging the rocket launcher and a player disappears the bot will unload its loaded rockets on his last seen position. Player might be caught in splash damage.
- For any weapon that causes splash damage, the bot will hold its fire if it means it could be killed by the splash.

For more detailed information check out the UT2004WeaponryShooting project at <svn://artemis.ms.mff.cuni.cz/pogamut/trunk/project/Addons/UT2004WeaponryShooting>.

- **Bots can now shoot any player.** To make use of the newly learned weapon skills the bot can now shoot any player using the shoot action. This means that link gun combos are possible now.
- **Improved navigation.** The pursue and goto action have been combined into a navigate action. This action will try to navigate to any navpoint or location on the map. When provided with a player, the navigation will follow this player. Individual percepts for lost and reached have been removed and replaced with a navigation state containing the current state of the navigation and its current destination.
- **Repeatedly calling navigation no longer blocks the bot.** Repeatedly calling the navigation (previously goto) with the same destination will no longer cause the bot the freeze in place unable to finish his path execution. Repeatedly calling the navigation with different locations will still cause the bot to freeze in place.
- **Three actions have been renamed.**
 - usePowerup(Powerup) to combo(Combo) to better reflect the terminology used by unreal
 - selectWeapon([Weapon]) to prefer([Weapon]), to indicate that the bot will merely prefer the weapon when available rather select it no matter what.
 - lookAt(Target)to look(Target), to simplify the syntax.
- **Four percepts have been renamed.**
 - status(IsShooting, Health, Armour, Adrenaline, Combo) to status (Health, Armour, Adrenaline, Combo), IsShooting has been moved to the current weapon percept, the Combo has been added to show the currently active powerup.

- `trackRecord(TotalKills, TotalDeaths, TotalSuicides)` to `score(TotalKills, TotalDeaths, TotalSuicides)` to reflect the terminology used by unreal.
 - `flagBase(Team, UnrealID)` to `base(Team, UnrealID)`, to simplify the syntax.
 - `gameInfo(Gametype, Map, GoalScore, TimeLimit)` to `game(Gametype, Map, GoalScore, TimeLimit)`, to simplify the syntax.
- **Specific navpoints are now visible.** To allow bots to predict when an item will respawn, the See Navpoint percept has been added. The percept is only provided for navpoints that have a pickup spot. When the navpoint is visible, but the item is not, the item has been taken and will respawn after some time.
 - **Memory consumption has been reduced.** Previous versions of the environment created a FloydWarshallMap for each agent. The current version implements a shared map and cache. This significantly reduces the amount of memory consumed by path planning.
 - **Visualizer Improvements.** The visualizer remembers the last position and size of all windows. A menu has been added to change the color settings.

9 Know Issues & Workarounds

A list of known issues and their workaround.

9.1 Window Focus in Windows 7

A known issue exists where when using Alt-Tab to switch to the UT2004 window it is possible that the window does not grab the mouse clicks. This can be resolved by holding down Alt-Tab and clicking on the UT2004 preview window with the mouse.

9.2 GOAL becomes non responsive on CTF-Colossus (and other huge maps)

Maps containing a ridiculous number of navigation points such as CTF-Colossus require a significant amount of memory to run path planning. By default the JVM only provides GOAL with 250MB. When GOAL runs out of memory it will lock up. You can avoid this by increasing the amount of available memory by starting GOAL with the `-Xmx512M` argument. For more information see <http://docs.oracle.com/javase/6/docs/technotes/tools/windows/java.html>