Charles University in Prague

EmohawkVille USER GUIDE

version 3.5.4

Artificial Minds for Intelligent Systems research group

Contents

1. Setup

There are three separate installations. The first is the EmohawkVille UDK game, which allows you to host server, join and play. The second is the Chef bot, which joins an EmohawkVille server and showcases possible bot behavior. The third is the Pogamut bot development environment (includes third party software).

1.1 EmohawkVille setup

EmohawkVille can be installed using the EmohawkVille installer available on the EmohawkVille homepage:

`http://pogamut.cuni.cz/main/tiki-index.php?page=EmohawkVille`

Please ensure your system satisfies the following minimum requirements:

- Windows XP SP3 (32-bit only), Windows Vista, or Windows 7
- 2.0+ GHz processor
- 2 GB system RAM
- SM3-compatible video card
- 3 GB free hard drive space

The installation contains several batch files. The "host_EmohawkVille.bat" and "host_hellsKitchen.bat" batch file start an EmohawkVille server with the respective map and displays its console. The "join_localhost.bat" batch file starts an EmohawkVille UDK client and attempts to join the EmohawkVille server running on the machine on the standard port. When you want to turn off or restart the server, please use the "kill_all.bat" batch file, as normal termination of the server console does unfortunately leave the server process intact.

When you join the server, press left mouse button to spawn your avatar. Please bear in mind that the hell's kitchen map contains over 400 items and thus the world initialization takes over a minute.

1.2 The Chef bot setup

The Chef bot can be downloaded from the EmohawkVille homepage. Please ensure your system satisfies the following minimum requirements:

- EmohawkVille
- Java SE 7 runtime
- 20 MB free hard drive space

Use the "run.bat" batch file to start the Chef bot. It will attempt to join the EmohawkVille server running on the machine on the standard port. A console will print out debugging messages including the currently executed task.

The bot will attempt to cook a predefined meal.

1.3 Pogamut bot development environment set-up

Pogamut bot development relies on a Maven enabled Java IDE as stated in the following minimum requirements:

- EmohawkVille
- JDK 1.7.0
- Maven enabled Java IDE (Eclipse Juno or Netbeans 7.3.1 recommended)

Once you have installed the IDE of your choice, create a new Maven project using the Chef bot archetype:

Archetype group ID:	cz.cuni.amis.pogamut.emohawk-ville.examples
Archetype artifact ID:	emohawk-ville-chef-bot-archetype
Archetype version:	3.5.4-SNAPSHOT
repository URL:	http://diana.ms.mff.cuni.cz:8081/artifactory/repo

Thanks to Maven, the required libraries should be downloaded automatically from the Pogamut artifactory (it may take a few minutes). The Chef bot can now be executed as a Java application with the following main class:

`cz.cuni.amis.pogamut.emohawk.examples.chefbot.EmohawkVilleChefBot`

You can now experiment with the Chef bot code, the original behavior can be easily suppressed by replacing the code in the `logic()` method of the `EmohawkChefBot` class.

2. World basics

Since EmohawkVille is a tool for experimentation with bots, some essential underlying concepts must be explained first.

2.1 Actor

Actor is an object in UDK. It can feature a position, a visual model, a collision model and can be replicated from server to client. Examples of an actor include a pawn (avatar), a cucumber, a navigation graph node or a wall.

2.2 Essence

Essences implement extended game mechanics of EmohawkVille. Each actor with extended game mechanics is linked with exactly one essence object. For example, cucumbers and pawns do have essences, navigation graph nodes and walls do not.

3. Human player in EmohawkVille

A human player can fully experience all features of EmohawkVille.

3.1 Controls

Controls follow the standard FPS scheme:

W	Forward
S	Back pedal
A	Strafe left
D	Strafe right
mouse	turn
E	Interact
Escape	Main menu

The interact button brings up the interaction menu. The menu is navigated as follows:

W	Up
S	Down
E	Select/Enter
Q	Cancel/Return/Close

3.2 World information

The user interface provides important information about the visible world in the following GUI elements: 1) crosshair, 2) rangefinder, 3) location display, 4) target actor name, 5) target essence info, 6) target essence attribute display, 7) avatar attribute display. The location of the elements is shown in Figure 3.1.

The crosshair is a targeting reticle in the center of the screen that denotes target. Most GUI elements display information about the target.

The rangefinder displays distance to the target location in Unreal units.

The location display shows coordinates of the targeted location. This is useful when debugging bots to compare the targeted location to a location reported by bot.

The target actor name shows the name as it is reported by the GameBots protocol to connected bots.

The target essence info displays the name of the essence linked to the targeted actor. It also displays its unique game object ID. The class of the essence can be easily discerned and matched to its Pogamut replica class.

The target essence attribute display shows attributes of the essence linked to the targeted actor. For reference attributes only a name is displayed.

Avatar attribute display shows attributes of the essence linked to the player's avatar actor.

Figure 3.1: Human player user interface.

4. World mechanics

This section of the guide explains general and specific world mechanics as well as game objects. It also explains how to perform various actions with a human player.

4.1 Actions and processes

An avatar can interact with the world through actions and processes. An action is an instant act, that either directly changes the world or initiates a process that carries out an operation over a period of time. Available actions can be found in the interaction menu (the E key). All actions have a limited range and the interaction menu shows only those actions that are possible.

When your avatar performs a process a special user interface is shown instead of the interaction menu. The Q key interrupts the currently executed process.

4.2 Items

Items are game objects that can either lay on the ground or be carried in an inventory of an avatar or be inside a container. Items can be picked up by pointing the crosshair at the item and opening the interaction menu.

The inventory is accessible from the interaction menu, it lists all currently possessed items and selecting one shows an interaction menu for that particular item. This allows for an item to be dropped from the inventory at the location pointed to by the crosshair.

Some items are countable and some are substances. Such items can be split and merged in the inventory.

4.3 Trade

Two avatars can exchange items. To initiate a trade, point the crosshair at the other avatar and open the interaction menu. Once the other avatar reciprocates, you will see the trade UI, which shows your inventory, the offer of the other avatar and the status of the trade. Item interaction menus can be used to add/remove items from your offer. You can accept the trade 2.5 seconds after the last change in offers. Once both sides accept, the items are exchanged.

4.4 Containers

Containers hold items. Items can be taken from and stored in a container by pointing the crosshair at the container, opening the interaction menu and selecting "Access container...". A container may disallow taking and storing of an item and this is reflected by the interaction menu ("can't store", "can't take").

4.5 Chopping board

Chopping board is used to chop choppable items. To do so, point the crosshair at the chopping board and open the interaction menu. Chopping requires possession of a kitchen knife (not a cutlery knife) and a precursor on the board. Chopping board is a container capable of holding at most one precursor-product pair.

4.6 Ingredients

Ingredients are items that can be eaten and most can be cooked.

4.7 Cookware and cooking stove

A cookware is either a cooking pot or a frying pan. It is a container item that can hold only ingredients. It has a limited capacity. When heated, the contained ingredients are cooked.

A cooking stove is used to heat up cookware. The power setting of each stove plate can be adjusted separately by pointing the crosshair at the cooking stove itself (not at a stove plate) and opening the interaction menu. Each stove plate is a container holding at most one cookware and it heats up the contained cookware.

A cookware on a stove plate may be interacted directly by some actions although it is technically in a container. This includes stirring for a cooking pot (requires a stirring spoon), flipping of ingredients for a frying pan (requires a turner), and accessing a cookware as a container. To access these actions, point the crosshair at the cookware and open the interaction menu.

The contents of a cookware can be served onto a dinning plate. To do so, select a cookware in your inventory while pointing the crosshair at the plate. This action requires a tool; a scoop for a cooking pot or a turner for a frying pan.

4.8 Cooking

Some ingredients can be cooked using various cooking procedures. Each cooking procedure defines a cooking quality represented by a float (0.0 - 1.0) attribute attached to the involved ingredient. The value of the attribute increases as the ingredient gains the respective quality.

Most ingredients can be charred (burnt). This is represented by the charredness quality and it happens when an ingredient is heated to high enough temperature for sufficiently long time. Stirring and the presence of cooking liquids (broth, water) reduce the chance of charring.

Some ingredients can be boiled. This is represented by the boiledness quality. To boil an ingredient, heat it up in a cookware above 70° Celsius. Add broth to accelerate boiling.

Some ingredients can be fried (both sides separately). This is represented by the friedness top/bottom quality. To fry an ingredient, heat it up in a cookware above 160° Celsius. Add cooking oil to accelerate frying. Flip the ingredient to fry it from the other side.

Broth is a unique ingredient. It is a result of adding water to a cookware or a dinning plate. Initially, it has a zero strength. Strength can be increased by heating up broth along with other ingredient above 70° Celsius. Cookware temperature is limited to 100° Celsius as long as broth is present, but the water will evaporate from the broth, increasing broth strenght and reducing broth volume as a result.

Table 4.1 shows behavior of individual ingredients or ingredient groups.

Ingredient	choppable	charring	boiling	frying	improves broth
Meat steak	yes	yes	yes	yes	yes
Meat bit	no	yes	yes	yes	yes
Whole vegetable	yes	yes	yes	no	yes
Vegetable slice	no	yes	yes	no	yes
Whole potato	yes	yes	yes	no	no
Potato slice	no	yes	yes	no	no
Broth	no	no	no	no	no
Cooking oil	no	no	no	no	no
Salt	no	no	no	no	no
Spaghetti	no	yes	yes	no	no

Table 4.1: Behavior of individual ingredients.

4.9 Dinning plate

A dinning plate is an ingredient container. Ingredients can be added either directly or served from a cookware. The contents of a dinning plate can be eaten by pointing the crosshair at the plate and opening the interaction menu. Eating requires a cutlery set (a spoon, a fork and a knife).

5. Pogamut API

The Pogamut client provides a bot controller class that exposes representation of the observed world, command interface and various helpful functionality, such as pathfinding. Bots should extend the bot controller class to gain access to the API (as is the case in the Chef bot).

The Pogamut client receives periodical snapshots of the observed world and creates an automatically refreshing representation. The command interface sends various commands to the UDK server that control the bot's avatar.

The bot code runs in `logic()` ticks that happen between snapshots and the observed world is guaranteed to remain unchanged during the execution of a `logic()` tick.

5.1 Replica

A `Replica` is a representation of an observed server-side object. Replicas often feature utility methods, for example, a replica of a container has the `canStore()` method to check whether or not an item can be stored in the container.

However, due to a technical limitation, replica may be torn-off between `logic()` ticks. A torn-off replica no longer receives updates. This happens mainly when the avatar loses sight of the replicated game object. For this reason, it is strongly discouraged to keep references to replicas between `logic()` ticks. Once a replica is torn-off, it will never be reattached again, instead a new replica is made if need be.

An essence is a type of a server-side object, that is linked to an actor as explained previously (2). Essences have an ID that is unique for the duration of the server execution. A replica of an essence receives this ID as well. Therefore, the ID can be used to identify a particular object. This can be applied to items, stoves and pawns (avatars), since all these game objects are essences.

Replicas can be accessed using the essence map:

```
getEssenceMap().getAllVisible( CLASS ).values()
```

This returns a list of essence replicas conforming to the passed class object that are linked to an actor. For example, an item contained in a container lacks physical manifestation of its own and thus is not linked to an actor and won't be listed.

5.2 Memorization

A memorization is a deep-copied state of a replica. Memorizations are not updated, as they refer to a past state. Memorizations keep the utility methods of the copied replicas.

The observation memory offers a method that performs the deep copy:

```
getObservationMemory().getMemorization( REPLICA )
```

Repeated calls with the same replica argument in the same `logic()` tick are optimized and return the same memorization. The observation memory can also look up the source replica of a memorization from the current `logic()` tick:

```
getObservationMemory().getPreimage( MEMORIZATION )
```

This method uses the type parameter defined by the `IObjectMemorization` interface to return as accurately typed replica as possible.

5.3 Sightings

The observation memory also maintains a list of sightings - the most recent observation of each essence as an independent object. Unfortunately, this does not include items in a container or an inventory. The sightings are saved as memorizations, so you can inspect the contents of a sighted container to circumvent the aforementioned limitation.

Sightings can be retrieved using:

```
getObservationMemory().getAllByMemorization( CLASS )
```

This returns a list of sightings conforming to the passed class object.

The observation memory forgets sightings when the essence is observed elsewhere or when the location of the sighting is observed empty. In other words, when the sighting is contradicted by a more recent observation.

5.4 Unifying interfaces

Memorizations and replicas of the same game object have a lot of common functionality and some concepts apply to both. For example, one can compute the total volume of ingredients in a cookware regardless of whether the cookware object is a replica or a memorization. For this reason a system of unifying interfaces exist. For example `IPotato` is implemented both by `PotatoReplica` and `PotatoMemorization`.

5.5 Foggy reference

Foggy references are slight nuisances caused by the partial observation of the world. This can be shown on the following non-EmohawkVille example: A kid is flying a kite, but the kid is obscured by a wall as shown in Figure 5.1. The kite object might have a reference to the kid, but the kid itself is not visible and thus not replicated. Such a reference is called a foggy reference. When accessing a foggy reference, the `isXHidden()` method must be checked first. If the referenced object is hidden, do not attempt to retrieve it.

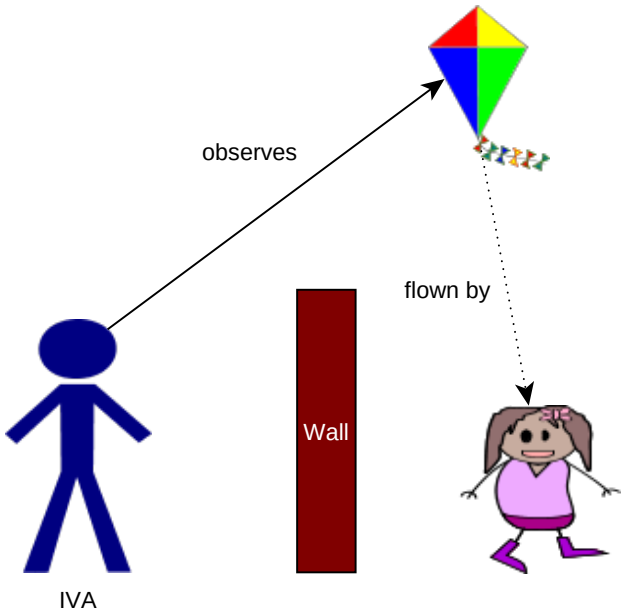

Figure 5.1: An example of a foggy reference. The bot observes a kite and the fact it is flown by someone, but does not see the child flying it.

5.6 Controlling the bot

Movement

The movement of an bot is a complex topic. Perhaps the best solution is to use the `GoToTask` of the Chef bot. This task is initialized by a reference to the bot itself, the destination and the proximity to the destination that is to be achieved.

Communication

A text message can be sent using the communication module:

```
getComm().sendGlobalTextMessage( TEXT )
```

Actions

Actions can be performed using the action registry:

```
getActionRegistry().getXAction().request( getPawn(), ... )
```

The action returned by a `getXAction()` call also has utility methods, most importantly it can tell the range of the action.

5.7 Finding replica/memorization class names

Names of concrete classes are directly derived from the Unreal script class names, which are visible in the game user interface. Interfaces are not visible this way, so a list follows for your convenience:

IObjectReplica A replica.

IEssence An essence.

IAction An action.

IPerformer An object capable of performing an action, currently implemented only by a pawn.

IProcess A process executed by a pawn.

IExecutor An object capable of executing of a process, currently implemented only by a pawn.

IItem An item.

ICountableItem A countable item.

ISubstanceItem A substance item.

IMergeableItem An item that can be merged, currently implemented only by countable and substance items.

IPossessor An object capable of possessing items, currently implemented only by containers and pawns.

IContainer A container.

ICollector An object capable of picking up items, currently implemented only by a pawn.

IBarterer An object capable of trading, currently implemented only by a pawn.

ICookware A cookware.

IIngredient An ingredient.

ICharrableIngredient An ingredient that can be charred (burnt).

IBoilableIngredient An ingredient that can be boiled.

IFriableIngredient An ingredient that can be fried (both sides separately).

IBrothableIngredient An ingredient that increases strength of broth.

6. Troubleshooting

6.1 Human player

Problem: I have successfully joined an EmohawkVille server as a human player, but I'm just a floating camera.

Solution: Press the left mouse button.

Problem: The action that I want to perform does not appear in the interaction menu.

Solution: Ensure you have met all the prerequisites, such as being within the range and having the required tools.

Problem: I can't add an ingredient to a cookware on a stove plate.

Solution: Ensure you are accessing the cookware and not the stove plate by selecting the correct option in the interaction menu.

Problem: I can't take a cookware from a stove plate.

Solution: Ensure you are accessing the stove plate and not the cookware by selecting the correct option in the interaction menu.

6.2 Pogamut client

Problem: I have the replica or the memorization of the object of interest that I want to remember between `logic()` ticks, what can I do?

Solution: The object is most likely an essence and therefore you can use the `getGameObjectId()` method either on its replica or memorization to get its ID. In the following ticks, you can find the current replica or a fresher memorization of the object by matching the ID.

Problem: My bot crashes on the assertion in the `FoggyRefReplica.get()` method.

Solution: You have most likely accessed a foggy reference without first checking the `isXHidden()` edge case. If that's not the case, please report the issue.