Getting started with AI Arena

Contents:

1. Downloading AI Arena
2. Running simulation
3. Controlling simulation
4. Modifying simulation data
5. Compiling source code
6. Creating custom AI behavior

**1. Downloading AI Arena**

AI Arena can be downloaded for free from [TU LINK DO DOWNLOADU]. One can download last stable release in simple ZIP package here: [LINK DO ZIPA], or clone git repository from here: [TU LINK DO REPO NA BITBUCKECIE].

After unpacking downloaded package or cloning repository, following directory structure can be found:

/

/build - working directory of AI Arena

/src - source directory for code and assets

/doc - documentation

/tools - useful tools for modifying AI Arena assets

**2. Running simulation**

AI Arena can be executed by using AI Arena Launcher in build/ directory - one can run it by double clicking build/Launcher\_cmd.jar file. Java runtime is required.

[TU PODSUMOWANIE CO MOZNA ZROBIC W LAUNCHERZE]

Alternatively, AI Arena can be run with simple command line scripts, also placed in build/ directory. When starting AI Arena with scripts, map to be started has to be selected by passing its name through -level parameter. For example, running map arena\_a can be achieved with following command executed inside build/ directory:

run\_args.cmd -level arena\_a

After starting AI Arena either way, user is asked for basic rendering parameters. One can change desired resolution, change mode to fullscreen, or disable antialiasing. Default values should be OK.

Remark: be sure to set "Floating-point mode:" option to "Consistent", if it has any other value. Simulation can run only with that setting.

Accepting selected rendering settings closes prompt window and starts simulation on selected level.

**3. Controlling simulation**

User controls actor in simulation world, called Player. Player has the same capabilities as AI actors - it can walk, run, jump, attack with melee or range.

Controls scheme for player are as follows:

- mouse movement - change looking direction; Player is observed through third person (TPP) camera

**- left mouse button** click - melee attack

- **right mouse button** click - ranged attack in direction pointed by camera

- **W** - move forward (with respect to camera orientation)

- **A** - move left (with respect to camera orientation)

- **S** - move back (with respect to camera orientation)

- **D** - move right (with respect to camera orientation)

- **space** - jump

- **shift** - sprint (use with **WSAD**)

One can also switch camera to 'freelook' mode, where camera isn't attached to player, and can move anywhere in the scene (player is constrained by collision and physics).

Switching to freelook mode is done with **F2** key, returning to attached camera - with **F3**.

Other features of simulation control:

- freeze simulation time - **F9**

- unfreeze simulation time - **F10**

- simulation speedup (10x) - **F11**

- simulation speedup off - **F12**

- show physics debug view - **F5**

- hide physics debug view - **Ctrl**+**F5**

- exit - **Esc**

**4. Modifying simulation data**

AI Arena enables simulation customization on following areas:

* scripting in Lua
* adding and modifying objects on levels by modifying simple JSON files or by sing specialized Blender plugin
* modifying AI behavior by creating new or changing existing actor controllers

Last area requires writing code, and is covered **in part 6. Creating custom AI behavior**.

**4.1 Lua Scripting**

Most of "game-specific" behaviors of interactive objects in AI Arena are written in Lua scripts, and therefore easily moddable. AI Arena offers simple scripting environment, where each simulated object has its own scripting context, where functions and variables can be defined.

By defining functions with specific names (later called "Script Handlers"), one can change behavior of game object. Script Handlers are called from native AI Arena code when some event happens, eg. when new frame is started, or object is created. Here follows list of available Script Handler types:

* onPostCreate - called once for object lifetime, after object is created
* onDestroy - called once for object lifetime, just before object is destroyed; please note object can be destroyed by using DestroyObject function, or by simply unloading the level (and that happens when simulation ends)
* onUpdate - called once per frame for every object; time intervals between every call vary, and can be obtained with GetTimeDelta function
* onTakeDamage - called when scripted game object takes damage from any object
* onTouched - called for objects of class VolumeTrigger (and subclasses), when other game object is inside its volume
* onExploded - called when object of class ExplodingObject explodes

Scripts used by AI Arena can be found in build/data/scripts directory.

4.2 Modifying levels

Levels description can be found in build/data/levels directory. Each level is defined in single .json file. Level format is simple - it's just collection of "game objects" - objects of classes implemented in AI Arena. For example, file:

{

"StaticCollidableMesh" :

{

"m\_meshName" : "arena\_a.mesh"

},

"PlayerSpawner" :

{

"m\_worldSpawnPos" : { 20.0, 2.0, 32.0 }

},

"AISpawner" :

{

"m\_spawnOrigin" : { -30.0, 2.0, -20.0 },

"m\_spawnRadius" : 1.0,

"m\_aiNum" : 1,

"m\_presetName" : "AISandboxBoss"

},

"CylinderVolumeTrigger" :

{

"Preset" : "lava\_lake\_r4",

"Name" : "lava\_mid\_lava\_1",

"m\_worldTransform.position" : { -4.0, 0.0, 4.0 }

}

}

Defines level with 3 game objects - one of class "StaticCollidableMesh", which is used for collision tests and rendering, another of class "PlayerSpawner", which is responsible for creating ("spawning") Player in specified position, and another of class "AISpawner", responsible for creating m\_aiNum=1 AIs of preset m\_presetName=AISandboxBoss in random places within radius m\_spawnRadius=1 meters around m\_spawnOrigin.

Last object also defines its name - "lava\_mid\_lava\_1" (objects can be referenced in Lua scripts by their names), and preset - "lava\_lake\_r4". Presets define default values for class fields - they're stored in data/presets directory, in JSON files, one per class. Preset files are similar to level files.

New game object can be added by simply adding new entry to level file.

**5. Compiling source code**

Source code can be found in src/code/mkd/ directory. Visual C++ 2008 or higher is required to build AI Arena from source. To do that, one must:

1. provide boost headers and compiled libraries; boost is not used in AI Arena source, but is required by OGRE rendering engine; instructions and download can be found at http://www.boost.org/
2. open AI Arena solution - src/code/mkd/mkd.vcproj
3. add boost headers and libraries to project or environment paths (*Tools->Options->Projects and Solutions->VC++ Directories*)
4. build mkd project

Besides boost, all other dependencies are included in package. AI Arena uses **OGRE**, **bullet**, **vjson** and **LuaJIT**.

**6. Creating custom AI behavior**

Class responsible for simulating AI-controlled actor is ActorAI. It exposes simple interface through which one can control it, with no boilerplate code needed.

To control ActorAI, one has to create its own ActorController - a class inheriting from IActorController interface. For real usage, see TestActorController and StateMachineActorController.

Steps:

1. Create your controller class in new files (it is important that your class name ends with "ActorController")
2. Inherit it from IActorController and override pure virtual methods
3. Include your class header file in src/code/mkd/ActorControllerFactory.cpp
4. Add new entry for your controller in ActorControllerFactory::create - if your class name is MyTestActorController, your entry should be CREATE\_CONTROLLER(MyTest)

**7. Nota bene**

1. Actor can be controlled by only one controller at a time
2. Controller class headers should not be included anywhere outside ActorControllerFactory
3. Controllers shouldn't be istantiated explicitly, just use factory
4. When adding new compilation unit (.cpp file) it is required to include "pch.h" before anything else
5. Controllers are not GameObjects, and do not use AI Arena RTTI - so they cannot be created on level and do not use presets