# EnAble Games Developer Libraries (EGDL)

Welcome to the enablegames libraries. The libraries make it possible to easily use a player's body as input to a game using our patent-pending System for Unified Input (SUKI).

The libraries also allow you to record motion and game info for remote monitoring, and use remote parameter setting for remote game adjustment.

## --EGDL performs the following tasks

1) Handles user login (eag\_Login scene)

2) Manages getting parameter overrides from server, setting parameters and saving/loading parameter profiles (eag\_MainMenu scene, DefaultParameters.json)

In the eag\_MoveCube scene:

3) Binds game variables to parameters (VariableHandler class, and egFloat, egInt, etc.)

4) Manages Avatar movement via network connection to enAblegames App (NetworkedSkeleton class)

5) Translates body motions to inputs (SUKI classes and SUKI files in StreamingAssets).

SUKI is the System for Unified Kinematic Input, which uses different .suki files to tell which body parts to monitor as input for the game.

6) Calculates joint angle data (SkeletonData)

7) Records game and body data to server (\_Tracker object and Tracker scripts)

In the eag\_KickCube scene, all but #5 are done.

# Section 1: Demos

## --Demos

There are two demo main scenes in the asset package at Assets/eag/Demos/Scenes: eag\_MoveCubeGame and eag\_KickCubeGame

eag\_MoveCubeGame: Move a cube controlled by a selected body part to collect falling coins. Uses SUKI.

eag\_KickCubeGame: Collect falling coins by kicking them with avatar's feet. Does not use SUKI.

There is also a vr version of the KickCube game which moves the camera to the avatar's eye position.

## --Running the Demos (QuickStart Guide)

1) Make sure you have an account on enablegames.com and have installed the enablegames launcher and any device drivers as detailed in

the Startup Guide at www.enablegames.com/support

2) Connect any trackers (i.e. Kinect) and Run the enablegames launcher (enablegames.exe) and log in.

3) Run unity, and create a new game project

4) Import the EGDL asset package (enablegames.unitypackage)

5) Load and run one of the two demos in Assets/eag/Demos/Scenes. Either eag\_Login or eag\_Login\_vr.

eag\_Login: PC version

If you want to build the demos, these scenes need to be added in the build settings

a) eag\_Login:calls eag\_MainMenu, eag\_MoveCubeGame, eag\_KickCubeGame

b) eag\_Login\_vr:calls eag\_MainMenu\_vr, eag\_KickCubeGame\_vr

## --Building and Running Demos on Android

- The android build will connect via wireless network to your computer running the enaAblegames launcher. The PC will pass the body tracking data to the phone as input to the game.

- Firewalls on the PC or port blocking on your network can prevent the launcher from connecting to your game. Please disable firewalls for testing.

1) Follow all Unity instructions for setting up your Android device and connect your device.

2) Load the vr scenes and add them to the project

3) Select Android in the File->Build Settings

4) In BuildSettings->Player Settings->Other, set:

a) Package Name to a unique name

b) Minimum API Level to Android 4.4: "Kit Kat", Level 19

c) If you want a stereo VR build, click the "Enable Virtual Reality" checkbox and add Cardboard and Daydream Virtual Reality SDKs

5) Click Build and Run button to build the .apk file, deploy it to your device, and run the app.

6) On the login screen, enter the IP address of the computer running the enablegames launcher. If the avatar does not show up, your game was unable to connect to the launcher.

# Section 2: Setting Up Your Project

## --Adding EGDL (QuickStart Guide)

**The basic steps to add EGDL to your game are:**

1. Load your game project.
2. Import the EGDL asset package (enablegames.unitypackage)
3. Add logging in and setting parameters support:
   1. Add eag/Demos/MoveCube/Scenes/eag\_Login and eag\_MainMenu scenes to your project build settings (these should be 0 and 1 levels)
   2. Set eag\_Login.\_EnableAPI object's Game and Version parameters to your game's name and version number.
   3. Set eag\_MainMenu.SceneManager object's GameSceneName to the name of your main game scene name (Play button loads it). Default is "eag\_MoveCubeGame".
4. Load your main game scene(s). This/these will be scene 2 and higher in the project build settings. If you are using any of the MoveCube or KickCube games, set the egGame.GameScene parameter to the next scene to load after the current scene is done. If you are using your own scene, open your scene and the MoveCube scene and then drag the egObjects object (which contains NetworkManager, \_Tracker, TrackingAvatar, and SukiSchemaList objects) from eag\_MoveCubeGame scene to your main game scene. These are the objects you need for input and tracking.   
     
   **IMPORTANT:** The MoveCube/KickCube scenes can be run without the Login or MainMenu scenes, and will use the DefaultParameters and won’t communicate with the server.

Graphical user interface

Description automatically generatedThe LoginMenu scene has an input for the IP Address field (which contains the IP address of the enAblegames Launcher/BodyTracker application), and it sets a PlayerPrefs variable with this address. This Network Address is also set in the egGame component (in MoveCube). If you don’t want this to override the value from the Login scene, you must comment out the setting of the PlayerPref in the egGame.Awake function:

**void Awake () {**

**PlayerPrefs.** **SetString(egParameterStrings.LAUNCHER\_ADDRESS, networkAddress);**

If you are running the launcher on the same machine as your game, you need to make sure it is set to *localhost, 127.0.01,*  or is blank.

1. Add the EG code to handle body motion input, tracking, etc. and modify to move your game object instead of Cube. Either add Game.cs or take code from it.
   1. Add code labeled starting with /// BEGIN ENABLEGAMES REQUIRED CODE from eag/Demos/MoveCube/Scripts/Game.cs script to your main game script which handles input.
   2. Call egAwake() and egStart() from your main game script's Awake and Start functions.
   3. Replace PlayerObject in egGetSukiInput() with whatever object or vector you want to be controlled by body input.
   4. Edit the footer data that will be written at end of a session (i.e. score, achievments) by changing the code in Footer.cs
   5. When level begins, call egBeginSession() to start recording the body motion data and game data to the server.
   6. When level ends, call egEndSession() to disconnect network client at end of level and write the footer data to the server.
   7. Add any parameters and variables you want to set in the parameter settings as described in the section below.
2. It is recommended that you set your game for the "Good" quality level in Edit->Project Settings->Quality so CPU usage will be balanced between your game and the launcher.
3. Edit the Build Settings->Player Settings to set your Company Name, and Product Name, Icon, and Splash Image Logo. (see examples in EGDL demo project).
4. After 1-7, you can run and build your game. To have the game parameters accessible via the portal, you will need to have your game and its DefaultParameters.json file added to the server and server code. This is done by sending it to the enAbleGames development team.
   1. The DefaultParameters.json file will be renamed your\_game\_name.json and be put at enableweb/server/games
   2. A 700x400 game image your\_game\_name.jpg will be put in enableweb/public/img/games
   3. Your game name added to enableweb/public/js/routes.js (and enableweb/server/test/api.settings.game.js for api testing)

## --Using Parameters

Parameters are values that can be changed by the player or remotely via the portal. These get bound to variables in the game to control difficulty, type of motion-input mapping, or even volume.

Graphical user interface, text, chat or text message

Description automatically generated

The EGDL stores all parameters in JSON files. The default file is called DefaultParameters.json and it is located in the StreamingAssets folder. This file is used to automatically generate the UI elements for each parameter in the MainMenu scene when the Settings button is pressed, including Sliders, Buttons, and Toggles. Parameters are also able to be overrided by a Clinic user via the enAblegames portal. These overrides are automatically handled by the EGDL.

All parameters can be Saved and Loaded to individually-named files for setting up a game for a particular user or type of therapy. This is also automatically handled in the MainMenu scene.

To modify a parameter's default value, simply edit the DefaltParameters.json file.

## Adding a new parameter:

To add a new parameter and bind it to a variable:

1) Copy an old one and change the name and give it a new unused $id for the parameter,

{

"min\_": 1.0,

"max\_": 10.0,

"value\_": 1.0,

"tick\_": 0.5,

"name": "Player Starting Speed",

"alias": "Player Starting Speed",

"description": "Starting speed for the player",

"$type": "RangeParameter",

"$id": "7"

}

2) Also add it to the bottom of the DefaultParameters.json file under the "parameters" entry, with the "$ref" the same as the $id number above

"Player Starting Speed": {

"$ref": "7"

},

3) In order to keep track of all the parameter names in one place and make sure they match in code and with the name in the DefaultParameter.json file, there are two classes to hold the names. The class ParameterStrings holds user-defined parameter names, and the egParameterStrings classs holds EGDL-required parameter names. Add your new parameter to the ParameterStrings.cs file:

public class ParameterStrings

{

public static string STARTING\_SPEED = "Player Starting Speed";

public static string GRAVITY = "Gravity";

}

4) Once a new parameter has been added, you can access its value by binding it to a variable via the VariableHandler class. The EGDL supports egVar variations of several native types, including egFloat, egInt, egBool, and egString. These can be used in place of their native types:

egFloat speed = 1.0f;

VariableHandler.Instance.Register(ParameterStrings.STARTING\_SPEED, Speed);

pos.x = pos.x + mov.x \* Speed;

The VariableHandler's Register function is passed the name of the parameter to bind to, and the egVar variable which is getting bound. If the variable is changed by the game, the parameter will be automatically updated, and if the parameter is changed, the variable will be updated.

**IMPORTANT**: Versions of the library prior to the June 23rd build: (<https://www.dropbox.com/s/5bfvzbkc39r06v1/Enablegames.dll?dl=0>)

require that you give a default value when declaring the egVars. For example:

egFloat speed = 1f; //GOOD – has default value

egFloat distance; //BAD – no default value

# Section 3: Adding Body Input (SUKI)

enAblegames translation of body inputs into game inputs is called SUKI (System for Unified Kinematic Input). SUKI loads external files which hold definitions on which body part to monitor and how to translate the body data to game input. These external SUKI profiles are stored as JSON files called Schemas, and are located in the SteamingAssets/SUKI directory.

## Default.suki

When you don’t select a suki profile in the Main Menu, it loads StreamingAssets/SUKI/default.suki which contains the names of all Suki Schema files to load.

Graphical user interface, text

Description automatically generated with medium confidence

IMPORTANT: SUKI profile files end in .**suki** extension. SUKI Schema files end in **.suki.json**  
In Unity’s Inspector, it doesn’t show the last extension. So the default.suki profile file shows up as default, and the schema file default.suki.json shows up as default.suki, which can be very confusing:

Actually, SUKI is given the name of a file with a .suki extension which holds the name of one or more JSON schema files. An example is LRSideBending.suki which contains the names of two schema files.

constraint.straightback.suki.json

torsosideleaning.suki.json

The first file is a SUKI constraint to monitor the spine and make sure the player does not lean too forward (slouch). The second file translates torso side-to-side bending into the game input. The JSON file defines the type of game input the body motion will translate into, such as a range between 0.0-1.0, and also the name of the input that the game will use to get the value, such as "joystick".

Typically, a user loads one of several SUKI files via the MainMenu parameters. The list of available SUKI profiles is in the parameters "Suki\_type" which contains the category of the file (subdirectory where it is located), and "Suki\_file" which contains the path and name of the file

{

"strings": [

"Arms",

"Balance and Reach"

],

"value\_": "Arms",

"name": "Suki type",

"alias": "Motion Tracking Category",

"description": "Category of motion tracking mode.",

"$type": "StringListParameter",

"$id": "3"

},

{

"strings": [

"default.suki",

"arms\\LElbowAngle.suki",

"arms\\LHandHorizontal.suki",

"balance and reach\\LReachWave.suki",

"balance and reach\\LRSideBending.suki",

],

"value\_": "default.suki",

"name": "Suki file",

"alias": "Motion Tracking Mode",

"description": "Which body motion to track to control the game.",

"$type": "StringListParameter",

"$id": "5"

},

--Using SUKI in your game

If you want to use existing SUKI files, there are several steps:

1) Add SukiSchemaList object (empty object with SukiSchemaList.cs script) to handle SUKI schemas

2) Use MainMenu Settings menu to load a SUKI file, or explicitly load one in code by calling SukiSchemaList.AddFile(string filename);

3) Add a TrackingAvatar to your scene at the origin. This is a Unity standard humanoid avatar which is the body that receives the body data from the enAblegames App. It must be at the origin so the correct position data is recorded to the server. If you need an avatar in a different location, you will create a second avatar that is ghosted to the tracking avatar. The TrackingAvatar has the following components:

Animator

Network Skeleton

Network Identity

Skeleton Data

TrackAvatar (see Recording Game Session below)

If you don't need to see the avatar in the game, we recommend disabling (unchecking) the SkinnedMeshRenderer component.

4) Add NetworkManager prefab to scene to connect to enAblegames App.

Adding a new SUKI file

Sometimes the existing SUKI files are not sufficient for your game,and you need to create some new ones. If you create/add a new SUKI file, there are several steps:

1) Add new .suki file (and any new .suki.json schema files) to the appropriate SUKI sub-directory

2) Edit the DefaultParameters.json file and add the new .suki name and path.

3) Add an associated animated GIF and PNG to StreamingAssets/suki\_icons directory motion for the MainMenu Settings' UI.

If you want a new stick-figure image and animation showing the body motion, use the program "Stykz" (stykz.net) to modify an existing one from the "SUKI icons" directory. ezgif.com can then be used to optimize the animated GIF size.

SUKI Names

Joint Names: used in "SchemaMetric": "SingleJointAngle" (1 joint)

namespace Suki

{

internal enum InputResolution

{

Trigger,

Signal,

Range,

Location2D,

Location3D

}

internal enum NodeMetric

{

SingleNodePosition,

SingleJointAngle,

MultiNodeVectorBetween,

MultiNodeAngleBetween,

SingleRobotAngle,

SingleBiometricValue,

}

internal enum Device

{

Kinect

}

//Bone Names (position point): used in "SchemaMetric": "MultiNodeVectorBetween" (2 points) or

// "SchemaMetric": "MultiNodeAngleBetween" (3 points)

public enum BodyNode

{

Hips,

Spine,

Chest,

Neck,

Head,

LeftUpperArm,

LeftLowerArm,

LeftHand,

LeftIndexProximal,

LeftIndexIntermediate,

LeftThumbProximal,

RightUpperArm,

RightLowerArm,

RightHand,

RightIndexProximal,

RightIndexIntermediate,

RightThumbProximal,

LeftUpperLeg,

LeftLowerLeg,

LeftFoot,

LeftToes,

RightUpperLeg,

RightLowerLeg,

RightFoot,

RightToes,

LeftShoulder,

RightShoulder

}

//Joint Names: used in "SchemaMetric": "SingleJointAngle" (1 joint)

public enum BodyAngle

{

Neck\_Flexion,

Neck\_LateralFlexion,

L\_Shoulder\_Flexion,

L\_Shoulder\_Abduction,

L\_Shoulder\_HorizontalAbduction,

L\_Elbow\_Flexion,

R\_Shoulder\_Flexion,

R\_Shoulder\_Abduction,

R\_Shoulder\_HorizontalAbduction,

R\_Elbow\_Flexion,

L\_Hip\_Flexion,

L\_Hip\_Abduction,

L\_Hip\_Rotation,

R\_Hip\_Flexion,

R\_Hip\_Abduction,

R\_Hip\_Rotation,

R\_Knee\_Flexion,

L\_Knee\_Flexion,

Spine\_Flexion,

Spine\_LateralFlexion,

Spine\_Rotation

}

internal enum Calculation

{

CrossProduct,

VectorAdd,

VectorMultiply

}

internal enum Reduction

{

Magnitude,

DotProduct,

XValue,

YValue,

ZValue,

}

internal enum Condition

{

GreaterThan,

GreatherThanEqual,

Equal,

LessThanEqual,

LessThan

}

}

## --RECORDING GAME SESSION TO PORTAL

The enAblegames' portal records all game session data to both a local file (in the <https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html>) and also the enAblegames Portal server. This includes the parameter settings for a game, all player motion data, any in-game data the game wants to specifically record, and a summary "footer" containing the game's score, duration, performance info, and even a player message. The EGDL Tracker classes handle recording of all data to the portal's server. In order to record information for your game, follow the following steps.

1) Add Tracker prefab which records game and body data to server (\_Tracker object and Tracker scripts)

2) Make sure the TrackAvatar component is on your TrackingAvatar.

3) Edit TrackerFooter.cs to grab and serialize any game data you want to show up on the portal under the game session info. FooterMessage fills in the values of all the serialed fields (variables preceeded with the [fsProperty] directive, e.g.:

[fsProperty]

public string playerNotes;