ENIGMA

A FREE, OPEN SOURCE & EDUCATIONAL GAME DEVELOPMENT KIT

Google Summer of Code

Steam Workshop SDK/Third Party Integration For ENIGMA

Contact Information

- Full Name (**Preferred**): Saif Salah El-Deen Yahya Mostafa Kandil (**Saif**)
- Primary Email:
- Secondary Email:
- Phone Number:
- Country (**City**):
- GitHub:
- Discord Username:

About ENIGMA

ENIGMA is an open-source 2D/3D game development software based on the popular YoYo Game's GMS (GameMaker Studio). ENIGMA uses a Drag & Drop system as well as its own programming language. EDL (ENIGMA Development Language) as it is normally called, is a mix between C++ and GML (Gamemaker Machine Language). EDL offers many powerful features and functions. Steam integration will extend such functionality as outlined in my proposal.

Abstract

I intend to address steam SDK integration #1881 idea. ENIGMA's project currently lacks third-party integration with steam. This idea is about creating a C++ wrapper for Steamworks API – which is developed and maintained by Valve corporation – so ENIGMA can use it. The project's final output will be something like this C# wrapper or this Java wrapper. Since ENIGMA supports function overloading – which is not supported in GMS – this wrapper will be specific for ENIGMA itself and cannot be used with any C++ application. This will be explained later. With the Steamworks extension, developers can integrate all steam features into their games when publishing on steam. These

features include a leaderboard, list of achievements, stats, cloud, and many more.

Here's a brief description of the main-goal APIs that will be implemented this summer:

- **General API**
  This API is for checking the availability of certain aspects of the steam client or server API. This means that these functions should be used before any other Steam API function call to ensure that the client/server setup is correct and communicating with the game.
- **Authentication and Ownership API**
  This API authenticates steam user's identity and verifies ownership of an application.
- **Overlay API**
  This API controls steam overlay.
- **Leaderboard API**
  This API supports persistent leaderboards with ordered entries. This API can display global/friend leaderboards in the game or community webpage.
- **Achievements and Stats API**
  This API makes it easy for developers to add achievements and statistics to the game.
- **Cloud API**
  This API is for saving players' progress upstream.
- **Social API**
  This API adds useful functions such as getting user's avatar.

Here's a brief description of the stretch-goal APIs:

- **Networking and Lobby APIs**
  Those APIs are for multiplayer sessions.

Steam SDK Compatibility with MSYS2/pacman

Background

Windows programs are normally built using MSVC and **.lib** library files. ENIGMA is/was primarily developed on Linux, where programs are normally built using GCC and **.a** library files. MinGW is a port of GCC to Windows that is mostly sort of compatible with MSVC libraries **but not entirely**. MSYS2 is a collection of libraries built using MinGW. pacman is the tool used to install MSYS2 packages, which ENIGMA uses mainly for managing its dependencies.

Solution

Steam SDK is not open source and doesn't have a license that allows redistributing. As result, steam SDK is not provided through **pacman** however it's provided through AUR **yay** – here – which ENIGMA uses as a 3rd party source. A package request has been provided for msys2/MinGW-packages #16412. I reached out to the MSYS2 Discord admins for the package, but no practical solutions were provided.

Documentation – as the one provided by GMS – for setting up Steamworks extension for ENIGMA will be provided for developers. This is a temporary solution for this issue until steam SDK is available through **pacman**.

I reached out to Steam support under reference code HT-WK2C-4R83-R3DV.

> Hi, the version of the Steamworks SDK that we distribute at https://partner.steamgames.com/downloads/steamworks_sdk.zip is the only one that we officially distribute. You can create wrappers but not redistribute copies of the Steamworks SDK. Separate versions for package managers may be convenient but we do not have the bandwidth to police them for including malware or being outdated.
>
> ...
>
> Steam Support
> Escalated Legal Review

ABI (Application Binary Interface) Compatibility

Background

An Application Binary Interface (ABI) is a set of rules and conventions that define how software components interact with each other at the binary level. An ABI defines basic data types, such as integers and floating-point numbers, their sizes, and alignment requirements. It also specifies how function calls are made, how parameters are passed and returned, and how exceptions are handled. Additionally, an ABI defines low-level details such as register usage, stack layout, and instruction encoding. One common ABI compatibility issue is binary incompatibility, which occurs when a library or object file is compiled with a different ABI than the application that uses it. This can result in errors such as segmentation faults, incorrect calculation of function parameters, or memory leaks.

Solution

Steam SDK is compiled using Microsoft MSVC, while ENIGMA uses MinGW GCC. This means we might face ABI compatibility linking issues. ABI compatibility issues are well-known since MinGW isn't fully compatible with MSVC yet. A **CMakeLists.txt** and **Makefile** files are provided for linking steam library to a game.

As steam SDK is not open source, we can't get a MinGW/GCC compiled library for it. We can't also compile it from scratch. As a result, it will be too annoying if we face ABI compatibility issues in the middle of summer. Till now all research done shows that we shouldn't need to use **libffi** with DLLs on Linux. This is because Libraries tailored to Windows use some Win32 macros. Those macros will force the correct calling convention in MSVC.

We've tested linking with modern C++ class definitions here.

In short, I expect that no ABI compatibility linking issues will be faced.

Implementation Plan

The next core functions will be implemented:

- `steam_init()`.
- `steam_shutdown()`.
- `steam_update()`. The developer must call this function once per render-frame. Example:

```
// EDL script code.
alarm[0] = 30; // if room speed is 30, steam_update() will be called every 1 second.
steam_update();
exit;
```

Note that any non-engine code must be placed inside `namespace enigma {}`, while any user code must be placed inside `namespace enigma_user {}`. User code is the user functions that the game developer can call in EDL scripts.

Main-goal APIs will be implemented in the following order:

1. General API

This API contains core functions of the Steamworks extension. The developer will need to perform some checks after initialising the API.

The following set of helper functions will be implemented:

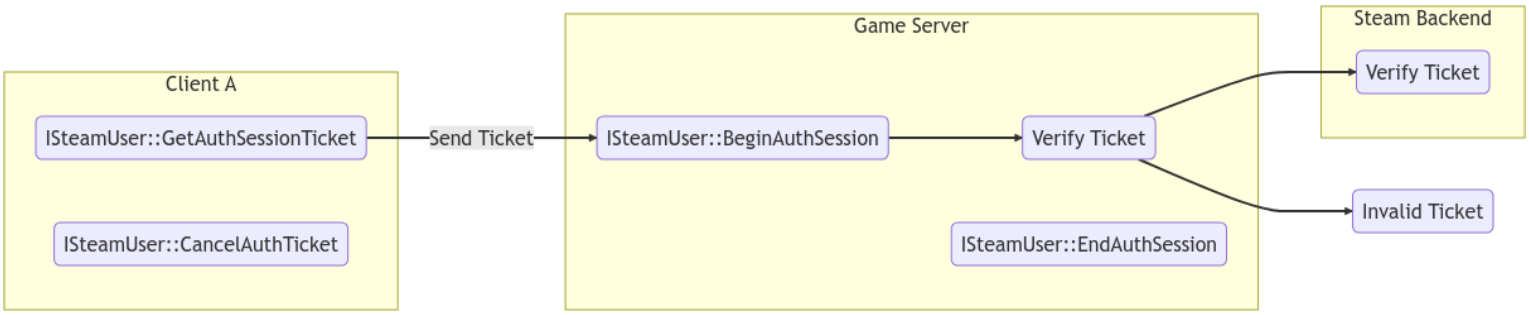
- `steam_initialised()`.
- `steam_is_user_logged_on()`.
- `steam_get_user_account_id()`.
- `steam_get_user_steam_id()`.
- `steam_get_user_persona_name()`.

`steam_set_warning_message_hook();` will be created for developers to debug the APIs while developing the game.

I will implement most of GMS General API functions as needed.

2. Authentication and Ownership API

Authentication API works the following way:



The next set of functions will be created:

- **steam_get_auth_session_ticket()**.
- **steam_begin_auth_session()**.
- **steam_end_auth_session()**.
- **steam_cancel_auth_ticket()**.

We may need to extend authentication functionalities for secure servers and encrypted tickets. This will be discussed after **May 4**.

There are stretch-goal APIs that will be implemented this summer or in another GSoC project. If main-goal APIs don't need extending authentication functionalities, I will ignore them. I will add those functionalities in the stretch period or in another GSoC project.

3. Overlay API

Steam overlay automatically hooks into any game launched from Steam. The overlay supports games that use DirectX 7 - 12, OpenGL, Metal, and Vulkan. ENIGMA supports both D3D and OpenGL on Windows (which will be used for this project).

The following set of functions will be implemented:

- **steam_is_overlay_enabled()**. The developer can use this function to synchronise between steam overlay and his game. Although the below

solution is inefficient, it's good for now until the new parser, called JDI (JustDefineIt), is done as it will support Callbacks. Example:

```
// EDL script code.
var is_overlay_enabled;
alarm[0] = 30; // if room speed is 30, steam_update() will be called every 1 second.
is_overlay_enabled = steam_is_overlay_enabled();
exit;
```

- **steam_activate_overlay()**.

We've written a SOG (Single Object Game) file which initialises steam API and then enables steam overlay using the following command and output:

```
saifs@Saif MINGW64 /d/enigma-dev
$ ./emake.exe -r CommandLine/testing/SimpleTests/steam_activate_overlay.sog -e
"GTest,Steamworks" -o /tmp/test.exe
```

```
SteamAPI_RestartAppIfNecessary completed successfully.

Setting breakpad minidump AppID = 480
SteamInternal_SetMinidumpSteamID:  Caching Steam ID : 76561199167979105[API loaded
no]
Pass : (is_initialized) == (1)
Pass : (is_overlay_enabled) == (1)
```

All the above prototypes are refactored so that I can use GTest on them.

4. Leaderboard API

Leaderboards are application specific and are set up on the Game Admin page of the Steamworks partner site, or via the API.

The next set of functions will be created:

- **steam_create_leaderboard()**.
- **steam_find_leaderboard()**. The developer can use this function to get the leaderboard he created on the admin page. Note that this function isn't provided by GMS's Steamworks extension.
- **steam_upload_score()**.
- **steam_download_score()**.
- **steam_download_friends_scores()**.

5. Achievements and Stats API

Achievements are set up on the App Admin page of the Steamworks partner site. The developer can request any achievement or stat from steam's backend. Steam will process the request asynchronously and provide a **Callback** when it's ready. This API will be implemented in two steps:

i. Overloading needed functions for achievement icons

- Investigating engine's protocol buffer format functions and how to handle sprite addition from RGBA image buffer.
- Overloading `buffers.h\buffer_create`, `GSsurface.h\surface_create`, `buffers.h\buffer_set_surface`, and `GSsurface.h\sprite_create_from_surface` functions.

I will do the same for profile avatar in Social API.

ii. Implementation

The following set of functions will be implemented in the following order:

- **`steam_request_current_stats()`**. This function must be called before any other.
- **`steam_set_achievement()`**. This function unlocks achievements.
- **`steam_get_achievement()`**.
- **`steam_set_stat()`**.
- **`steam_get_stat()`**.
- **`steam_store_stats()`**. This function must be called after editing any stat or achievement.

The next set of functions will be implemented for debugging purposes:

- **`steam_clear_achievement()`**.
- **`steam_reset_all_stats()`**.
- **`steam_reset_all_stats_achievements()`**.

Debugging functions are stretch-goal. I can just stick to steam console by running `achievement_clear <appid> <achievement name>` and `reset_all_stats <appid>` which will clear all data for a specific game.

6. Cloud API

Steam Cloud automatically stores files from your game on Steam's servers. This means that players can log into Steam and access their saved games from any computer.

The next set of functions will be implemented:

- **steam_is_cloud_enabled_for_app().**
- **steam_is_cloud_enabled_for_account().**
- **steam_file_exists().**
- **steam_file_size().**
- **steam_file_persisted().**
- **steam_file_write().**
- **steam_file_write_file().**
- **steam_file_read().**
- **steam_file_share().**
- **steam_file_delete().**

You can look at Steamworks extension's prototype and how I will organize project structure in my fork here. Note that the actual implementation of all provided prototypes will be different.

Debugging

Steam offers very useful ways for debugging Steamworks API for bugs. `C:\Program Files(x86)\Steam` and `C:\Program Files(x86)\Steam\bin` are added to environment variable. Running **steam.exe -console** will open steam with console commands for tracking the API. Adding **-debug_steamapi** for using `ISteamUtils::SetWarningMessageHook` for human-readable error messages.

The debugging output will look like this:

```
Calls over the last 46167 milliseconds:
Process                 Method                                  Calls      First      Last
----------------------- --------------------------------------- ---------- ---------- -------
steamworks.exe          IClientApps::GetAppData                 1          17515      17515
steamworks.exe          IClientUtils::GetSteamEnvironmentForApp 2          17515      17518
steamworks.exe          IClientUtils::GetAppID                  3          17521      17552
steamworks.exe          IClientUtils::RecordSteamInterfaceCreati 2         17522      17552
steamworks.exe          IClientUser::GetSteamID                 1          17553      17553
```

Testing

Check project structure for testing our changes with LGM here.

There are two more files that must be placed in the same directory as the executable. LGM's run button places the executable inside `/msys64/tmp/` on Windows. The **steam_api64.dll** file will be linked dynamically to the executable. The **steam_appid.txt** contains **appID** for our game.

Since my build system's PR **#2334** isn't finished yet, I've provided a **Makefile** file for linking.

I will run our game from CMD using the following command:

```
saifs@Saif MINGW64 /d/enigma-dev
$ ./emake.exe ../sample_game/sample_game.gmx/sample_game.project.gmx -e
"Alarms,Paths,Steamworks" -o /tmp/test.exe
```

When running dumpbin /dependents /msys64/tmp/test.exe from Developer CMD VS 2022. Our **DLL** linked dynamically as we expect:

```
Dump of file C:\msys64\tmp\test.exe
File Type: EXECUTABLE IMAGE
Image has the following dependencies:
steam_api64.dll
ADVAPI32.dll
COMCTL32.dll
comdlg32.dll
DSOUND.dll
GDI32.dll
KERNEL32.dll
msvcrt.dll
ole32.dll
OPENGL32.dll
SHELL32.dll
USER32.dll
WININET.dll
WINMM.dll
```

Unit testing

- Unit tests will be written using GTest library and placed in `/enigma-dev/CommandLine/emake-tests/Extension/Steamworks/`.
- SOG (Single Object Game) files will be written and placed in `/enigma-dev/CommandLine/testing/SimpleTests/`.

Deliverables

- C++ wrapper for Steamworks API so it can be used by ENIGMA as an extension such as the one provided by GMS (**Required**).
- An example game to showcase the capabilities of the extension such as the one provided by GMS (**Required**).
- Steamworks extension setup documentation for game developers (**Required**).
- A published game on steam that uses all the implemented APIs (**Optional**).

Timeline

- **Before May 4:**
  - Investigating more EDL (ENIGMA Development Language) scripts.
  - Investigating Steamworks API.
  - Finishing my PR #2335.
  - Investigating codebase for engine's protocol buffer functions.
  - Getting schoolwork done so I have less work after **May 4**.

- **May 5 – May 28 (before the official coding time):**
  - Set up regular communication channels with mentors.
  - Read ENIGMA's code of conduct.
  - Getting more familiar with Steamworks API by investigating the docs more carefully.

- Building the example game which will be delivered with the extension.
- Discuss project goals, milestones, and deliverables with mentors.
- Discussing final project structure with the mentor.
- Discussing changes to the engine's protocol buffer format.
- Implementing General API.
- This is a great period for getting schoolwork done. Having a lot of schoolwork in parallel with both final exams and GSoC project after **May 28** is not good (check my commitments).

- **May 29 – June 10 (Official coding period starts) (Phase 1):**
  - Finishing General API (if it's not done before **May 28**).
  - Implementing Authentication and Ownership API.
  - Writing unit tests for all the implemented functions.
  - Document all the implemented functions.

- **June 11 – June 23:**
  - Implementing Overlay API.
  - Implementing Leaderboard API.
  - Writing unit tests for all the implemented functions.
  - Document all the implemented functions.

- **June 24 – July 5:**
  - Investigating engine's protocol buffer format functions.
  - Overloading engine's protocol buffer format functions. This makes this C++ wrapper specific for ENIGMA only.
  - Implementing Steam Achievements and Stats API.
  - Writing unit tests for all the implemented functions.
  - Document all the implemented functions.

- **July 6 – July 9:**
    - Updating/Fixing/Improving on mentor's call.
    - Writing more unit tests and SOG files.
    - Updating/Fixing/Improving documentation written.

- **July 10 – July 13 (mid-term evaluations):**
    - Submitting PR.
    - Documentation.
    - Adding the written documentation to ENIGMA's WIKI.

- **July 14 – August 10 (standard coding period):**
    - Implementing Cloud API.
    - As listed on GSoC official website, first evaluation will be on **July 14**, so I think that's a perfect period for getting an **appID** for publishing our game. Getting an **appID** will require some fees and time.
    - Publishing a game to steam and trying to test all the implemented APIs (if Game Developer or Publisher profile is done).
    - Writing unit tests for all the implemented functions.
    - Document all the implemented functions.

- **August 11 – August 22 (standard coding period):**
    - Implementing Social API.
    - Writing unit tests for all the implemented functions.
    - Document all the implemented functions.
    - Improving ENIGMA's WIKI.

- **August 23 – August 27 (standard coding period):**
    - Continue coding unimplemented functions (for unexpected latency).
    - Pushing changes to the PR and finalizing it.
    - Documentation (if needed).
    - More unit tests (if needed).
    - Creating setup instructions – as the one provided by GMS – for developers as steam SDK isn't compatible with MSYS2/pacman yet.
    - Submit the demo game published on steam with all implemented and tested functionalities.

- **August 28 – September 4 (final evaluations):**
    - Continue coding unimplemented functions (for unexpected latency).
    - Documentation (if needed).
    - More unit tests (if needed).
    - Review the implemented APIs for enhancements.
    - Ask my mentor to extend my period for any unexpected latency.

The rest of the timeline is for any unexpected latency.

If all the required APIs are implemented successfully in time, I will extend the period for the rest of the APIs.

- **September 4 – October 31 (extended period):**
    - Continue coding and finished the rest of the functions and APIs.

The rest of the bullet points are stretch-goal:

- Implementing Networking and Lobby APIs.
- Writing unit tests for all the implemented functions.
- Document all the implemented functions.

- **November 1 – November 6 (final evaluations for all GSoC contributors):**
    - Submitting and finalizing PR.

The rest of the bullet points are a stretch-goal:

- Documentation.
- Writing more unit tests and SOG files.

- **November 7 – November 13 (final evaluations for all GSoC mentors)**
    - Keeping in touch with mentors for any updates.
    - Documentation.
    - Review the implemented APIs for enhancements.

My Contribution to Open-Source Projects

a) My Contribution to ENIGMA

b) My Contribution to Other Projects

Additional Commitments

Commitment	From	To	Hours/Week (Maximum)
Final Exams	**27 May, 2023**	**15 June, 2023**	12

My Familiarity with ENIGMA

Although I got rejected in last year's GSoC with ENIGMA, working on ENIGMA's build system was very beneficial. I got the chance to work on ENIGMA's engine which contains all the different ENGIMA systems including the extension system which is the core of steam SDK integration project. I'm familiar with most of the codebase.

My Participation in the Google Summer of Code Project with the ENIGMA Project

- I agree to be in contact with my mentor or project administrator at least once a week.
- I agree to attend the meetings, webinars, and other possible online events for students and mentors as many of these as possible given my time zone.
- I agree to comply with the code of conduct of ENIGMA.
- I agree to maintain Steamworks ENIGMA's extension for any new version of steam SDK or any changes to ENIGMA such as upgrading the engine's build system **#2334**.