GGE Manual

Daniel Johansson

April 28, 2022

Contents

Part I
GGE in short

GGE stands for Grid Game Engine. It is inteded to be used as a game engine for grid based games such as chess or Sid Meier's Civilization. It is not only restricted to turn based games.

The engine itself is written in C/C++ and depends on SDL2. Games for the engine are written in Chai script, a C++-header scripting language.

There are tools for generating code written in Perl5 though they are not necessary for using the engine.

The following two chapters will cover the inner and outer workings of the engine.

The inner working of the engine is not strictly necessary to know when writing a game but it may help understand what happens under the hood. This chapter should be helpful in case of modifying the engine or adding custom game engine modules.

The outer working of the engine is the (chai) scripting part of it. This is necessary for understanding the processes of making a game. It will also thuroughly describe the example "game" provided.

Part II
Inner Workings

1 The Core

At the core of GGE is the Core object. The core in of itself does not do much in the way of running a game, for that modules are used such as the graphics module. However, without the core none of the many parts of GGE would be able to function.

2 Chai - GGE Communication

Communication between the game, written in Chai script, and GGE is done via the `Scripter` module. This is done by exposing the `GGE_API` object to the chai script which in turn communicates with the rest of GGE.

3 Modules

GGE performs anything but the bare essentials through modules. An example of a module is the graphics module. It handles all the graphics related functionality of the engine such as rendering graphics on the screen.

If the game does not initializes any modules nothing can happen. If the game does not add any commands (see 6) for initalized modules nothing will happen. You will not even be able to quit the game since no event module that handles input exists. In order for the game to make use of a module in the game engine

1. Initialize the module to be used.
2. Add a command for the module if necessary.

The standard modules that can be used. Internal module name and class names are only used within the engine. In the script the external module name is used, sometimes as a suffix (e.g. `init_events`).

Internal Module Name	Class Name	External Module Name
GRAPHICS	`Graphics`	graphics
EVENTS	`Events`	events
GRID	`Hex_grid`	grid
SCROLLER	`Scroller`	scroller
TEXTER	`Texter`	texter

3.1 Module Handling

3.2 Registering a Module

In order for GGE to run modules they must be registerd in the file `registerd_gge_modules.hpp`. Next modules must be initializable, either via another module's initialization function or its own initialization function. These functions should reside in the `GGE_API` for exposure outside.

4 Generating Code

Perl5 is used to generate a bunch of code. This is not necessary in order to run the engine. While not strictly necessary to use when modding the engine there are unfortunately some places of the code that require repetitive coding which can be skipped with the scripts in `build_tools/`. Simply run all scripts ending in `.pl` and all necessary code will be generated. The table below describes the different scripts and their functions.

Script Name	Writes to	Description
`expand_gge_module.pl`	`gge_module.hpp`	Generates cases for stringifying a modul
`expand_modules.pl`	`moduler.hpp` and `moduler.cpp`	Generates getters and setters for mod
`expand_modules.pl`	`moduler.hpp` and `moduler.cpp`	Generates getters and setters for modules in the Moduler objec
`generate_commands.pl`	`commands/command.hpp` and `commands/<specific_command>.hpp`	Generates command enum in the `Command` parent class based on exported functions i
`generate_get_gge_modules.pl`	`generated_get_gge_modules`	DEPRICATED
`generate_gge_api_defaults.pl`	`gge_api_defaults.generated`	Generates template code for exposing functions in the

Note: the Perl scripts to not check for errors nor always generate nicely formated code. Make sure your code is correct or finding erros will be a hassle.

Part III
Outer Workings

5 Initializing a Module

Modules are initialized through the `gge_api` functions beginning with `init_` e.g. `init_graphics(<ARGUMENTS>)`.

6 Adding a Command

Once modules have been initiated commands can be added to the module through the `gge_api` function `add_command` The argument is a string in the form of `<CLASS_NAME>.<CLASS_FUNCTION>(ARGUMENT)`.
Example: `gge_api.add_command("graphics.draw(grid)")`