hardLizard0 Engine

John Emory,
hardLizard Studios

***Abstract*—The hardLizard0 Engine is a Game Engine written in golang targeting lower end hardware with tile based games. The engine uses ebiten as a rendering backend. The engine implements an Entity-Component-Systems (ECS) framework to achieve better performance on lower end hardware.**

I. Introduction

THE hardLizard0 Engine uses an ECS framework to provide a framework for tile based games on lower end hardware. The Entity-Component-Systems (ECS) framework allows the engine to prioritize cache coherency in the CPU, avoiding cache misses. The use of tiles and spritesheets for graphics minimizes graphics memory used while capturing a retro aesthetic. The rendering engine Ebiten is used to handle video output, player input, and audio. For the hardLizard0 engine, minimal custom tooling for designers is to be developed in order to prioritize engine development. Tiled is the preferred level design tool.

July 13, 2019

II. Design

Explain what you are designing because at this point the reader doesn't know what you're talking about yet. Explain what you're making and how you intend to achieve it. Under every section heading you MUST say what subsections are contained within it, think of it as a baby introduction. Write the subheadings first tho.

An ECS framework is an alternative approach to the usual object oriented programing (OOP) paradigm often used for game engines. Instead of creating and destroying objects that may be fragmented in memory, an ECS framework consists of Entities, Components and Systems. Entities are restricted to unique Global Identifiers (GIDs) and hold no state of their own. Instead, a GID refers to the index of any of several arrays, called Components. A Component is an array that holds information that is common between entities. A System is a function that iterates over one or several Components, updating the global state that the Components constitute.

A. Inspirations

Drawing from the 16-bit aesthetic of the SNES, hardLizard0 aims to recapture this retro-aesthetic while simultaneously lifting some of the restrictions imposed by the state of the art in the 1990s. Even the most basic hardware still in use today is orders of magnitude more powerful than the early 1990s video game consoles. By lifting these restrictions, we can allow greater artistic breadth for game designers while retaining some of the architectural decisions that allowed that generation of games to be both performative and visually appealing.

B. Goals

The aim of the hardLizard0 engine is to provide a platform on which to build tile based games for modern, if lower powered, hardware.

C. Software Design

The use of an ECS framework ensures the engine will be maximally performative on minimal hardware. A pitfall of OOP, though it may not be important in every program, is memory fragmentation. When creating and destroying objects in memory, one does not have memory cohesion between common fields between objects. Although an ECS framework increases design complexity of the engine, the potential performance gains, allowing less substantial hardware to run the engine, more than offsets the cost of complexity of design.

III. Mechanics

The hardLizard0 Engine aims to be a versatile engine that allows a game designer to realize a vision of a tile based game, irrespective of genera. Included target generas are:

- Platformer
- Metroidvania
- ActionAdventure
- Shoot'em-up
- Puzzle

The engine will be flexible enough to accommodate toggling gravity and different layer collisions. A companion program for compiling json levels and png tilesheets and spritesheets into go files to be compiled into the final binary. This will simplify deployment of binaries for various platforms.

A. Level design

For the preliminary version of the hardLizard Engine, the Tiled **insert link** map editor will be the prefered level editor. The level compiler will accept world JSON files created by Tiled, supporting the following feature:

- maps
- layers
- layer offsets
- tilesheets
- spritesheets
- animated tiles
- spawn points
- hitboxes
- map transitions

IV. Development

Initial work on a Tiled JSON loader has been completed. For better performance, a custom tool to compile JSON into a go file will be needed. Many Components have already been implemented. An ongoing list of features is maintained at https://github.com/hardLizard/hardLizard0.

V. Pipeline

Explain how the subs from DEVELOPMENT will be integrated, not how you'll code them but how you'll tie them in and the order in which you'll do them.

VI. Asset Development

A common set of menu borders and rendered text should be provided for game designers to use, unifying a system aesthetic.

VII. Analysis

Future versions of the engine should include support for simple 3D-graphics. Moving variables out of global variables while still avoiding heap allocation, or chosing a different language for development would be good to simplify retooling the main engine to fit a given project.

VIII. Conclusion

You should also learn how to do references and use literature to substantiate decisions as well as literature that challenges your decisions.

DRAFT DRAFT DRAFT DRAFTDRAFT DRAFTDRAFT DRAFTDRAFT DRAFT