UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

OpenGL Framework

author

Dragoş-Andrei Bobu

session: iunie, 2024

scientific coordinator

Conf. Dr. Varlan Cosmin

UNIVERSITATEA ″ALEXANDRU-IOAN CUZA″ DIN IAȘI

FACULTATEA DE INFORMATICĂ

OpenGL Framework

Dragoş-Andrei Bobu

Sesiunea: iunie, 2024

Coordonator științific

Conf. Dr. Varlan Cosmin

Abstract

The developed application features an intuitive OpenGL interface that allows users to interact with vectorial math operations, render graphics primitives, and manage game objects. The interface is designed to be user-friendly, catering to both beginners and advanced users.

Contents

Part I

Overview

″University teaches you more than what to think. It teaches you how to think″
Stefan Pantiru

Introduction

What you are about to read is over 5 years of game engines research combined with my recently gained machine learning experience aquired as part of my bachelors programme.

Each part acts as an introduction to the one following it. You are now reading about the academic influence that lead to this extended study.

Of course, machine learning wasn't the only knowledge i gained during my bachelors study. I also take pride of myself for actively improving my Software Engineering and Object-Oriented-Programming knowledge, my System Design and Management skills and Computer Graphics and Computational Geometry understanding.

Besides the knowledges, i have gained much experience working with industry-standard tools. There were courses where the main goal was a specific programming language understanding. Courses like Python Programming, OOP, Python Programming, PLP and many many more. Courses like this made me an agile handyman that has his toolbox in order.

Motivation

In the previous i mentioned the toolbox that this university allowed me to organise for myself. In the following i will go over some of the tools i specifically tailered for this implementation.

The biggest motivator factor for persuing this project is a strong personal interest in the field of game development and computer graphics.

My goal became to apply some of those concepts aquired during my study in my personal research.

The way i chose to apply these concepts is by building a graphics framework compatible with some of the simpler machine learning solutions.

Goals

The goal is to build an Abstraction Layer that opens the inner workings of computer graphics and machine learning.

This Abstraction Layer **must** respect PHIGS standards, **must** be easy-to-learn for both intermediate and beginner end-users, **must** present shorter and more straightforward solution than existing solutions.

The project's theme is combining Machine Learning with the Game Makers' Environments and standards.

In the following i will present the benefits of this theme.

Game Engines

Building your own game engine is the standard practice when it comes to large in-house development teams. This of course, comes with both advantages and disadvantages. The advantages are spaced around the ideas of security and integrity. While the disadvantages are mostly cost-related.

Smaller companies often turn to open-source software as a solution. There are several providers in the market, such as Unity, Unreal, and Godot. Other notable names include P5.js, Processing, and even content creators like The Coding Train, Sebastian Lague, and Code Bullet, as well as electronics expert Ben Eater.

If you're not familiar with these references, don't worry. The following sections will provide more in-depth insights into each of them.

Overview

As I have already previously mentioned, this is a standardized practice that seems to return profit on investment. Large part of the returnment is based around security, bussines integrity and competitive edge.

Not many of those closed-source software had ever seen the light of end-users' eyes. Some get leaked sometimes. (Figure:)

On the other side of the spectrum live the restricted source software and the open sourse software.

In the following i will like us to analyse some of the following game engine environments and draw out relevant particularities

Machine Learning

Machine Learning is a fun/handy technology to integrate.

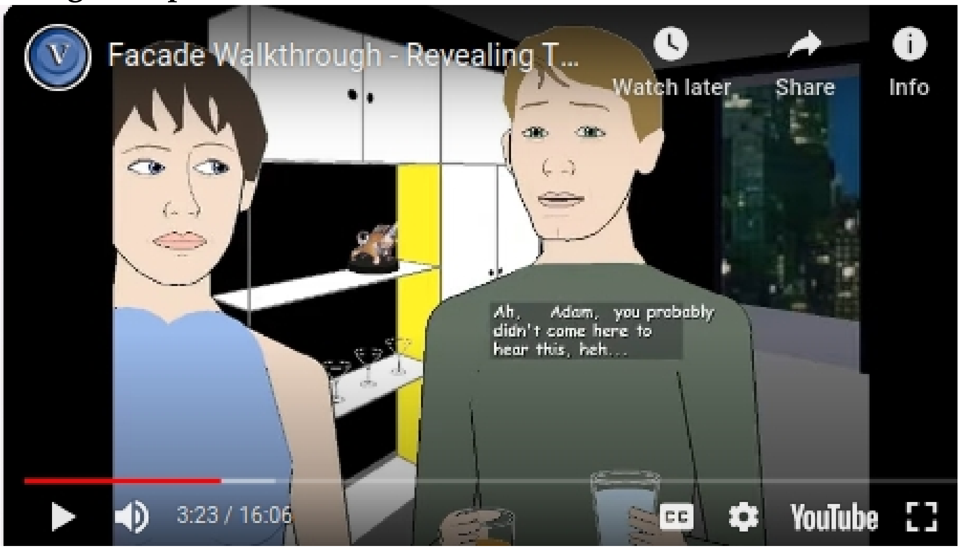
is eye candy, everyone implements it into anything. Let's see how they did it.

What other reasons could there be for wanting to implement it?

Dialogue is important in simulators.

In game design, it's a known fact that the npcs are there to guide the player, are the pawns in a chess game. They are an immovable objects that determines the max speed of the unstoppable force.

This static elements could be greatly improved. **Not only gamers would benefit from dialogue improvements!**


Facade Walkthrough - Revealing T...
Watch later
Share
Info
Ah, Adam, you probably didn't come here to hear this, heh...
3:23 / 16:06
YouTube

NPCs are there to guide the player and are the projection of the game designers into the game world.

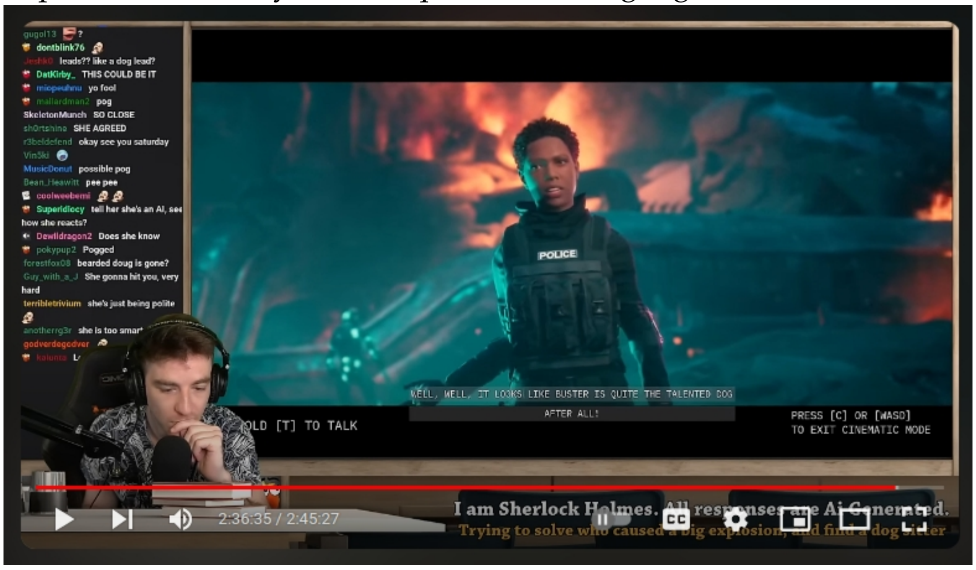
Because of this, is is really important that npcs have fluid dialogue and dont break the illusion of choice too easily. Current solutions imply using dialogue trees. Dialogue trees are an really good solution but they can still feel rough on the edges. and the illusion can be broken easily when u have to decide from a set of predefined dialogue choices.

The imersiveness of games could greatly improve if ml were to be implemented on top of this already existing dialogue tree solution.

Such solutions have already been experimented with, in the following i will present the findings of 2 other papers that use machine learning to improve npc dialogue and interaction.

Simulating Human Interaction

"Inworld Origins is a technical demo developed to demonstrate the power of generative AI-powered NPCs in video games. It showcases advanced NPC behavior and dialogue driven by artificial intelligence, leading to more immersive and personalized gaming experiences. The demo is set in a neo-noir sci-fi crime scene in the aftermath of an explosion in the city of Metropolis." source:google



gugol13 ?
dontblink76
Jeshk0 leads?? like a dog lead?
DatKirby_ THIS COULD BE IT
miopeuhnu yo fool
mallardman2 pog
SkeletonMunch SO CLOSE
sh0rtshine SHE AGREED
r3beldefend okay see you saturday
Vin5ki
MusicDonut possible pog
Bean_Heawitt pee pee
coolweebemi
Superidiocy tell her she's an AI, see how she reacts?
Dewildragon2 Does she know
pokypup2 Pogged
forestfox08 bearded doug is gone?
Guy_with_a_J She gonna hit you, very hard
terribletrivium she's just being polite
anotherrg3r she is too smart
godverdegodver
kalunta L

WELL, WELL, IT LOOKS LIKE BUSTER IS QUITE THE TALENTED DOG AFTER ALL!

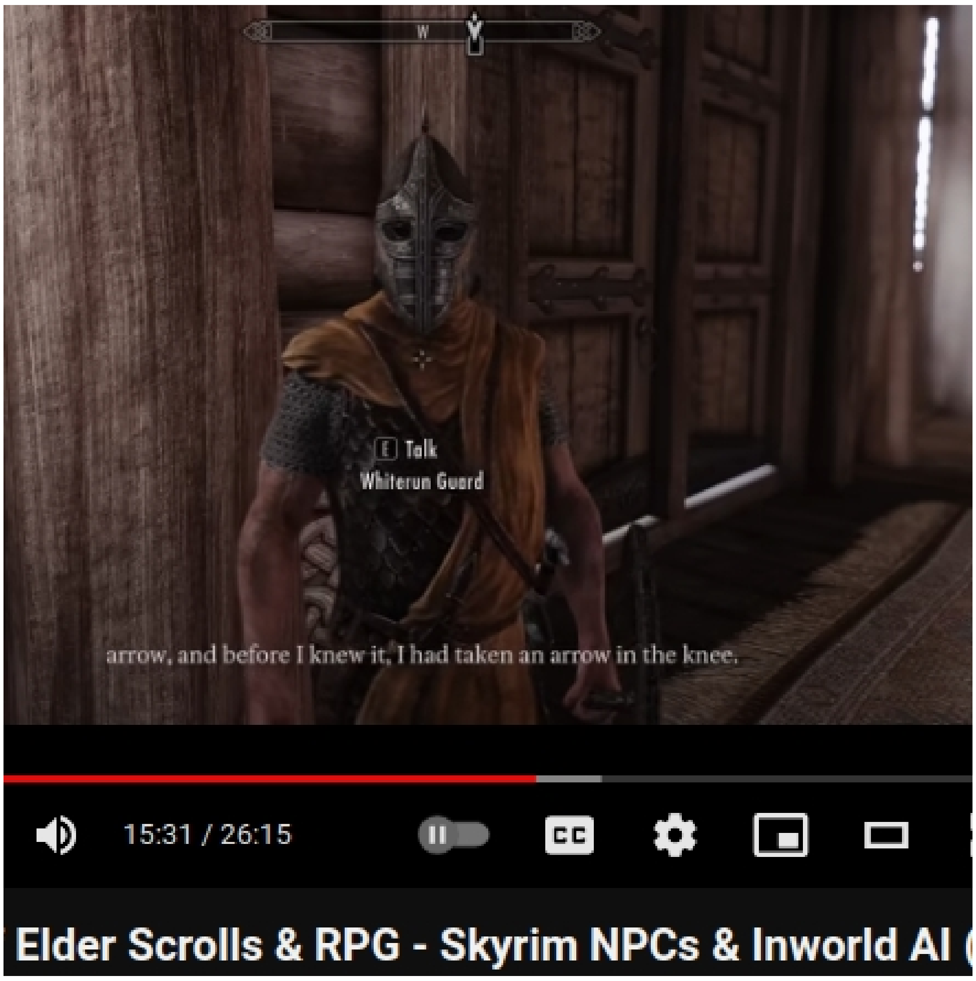
I am Sherlock Holmes. All responses are Ai Generated.
Trying to solve who caused a big explosion, and find a dog sitter

Screenshots of one demo using their openai abstraction layer.

This team even offers multiple solutions for implementing such agents in popular environments such as Unreal.

Something that this project and the following one have in common is the presence of microphone input for live dialogue.

There is also a mod for the popular game Skyrim that allows the player to have fluid dialogue with any in-game character.

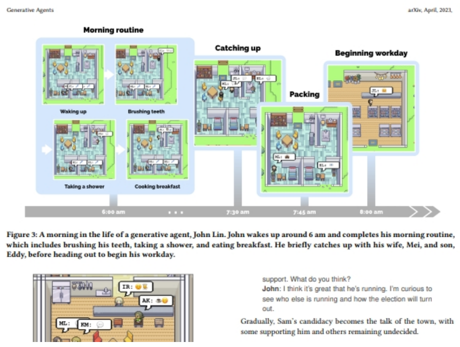


arrow, and before I knew it, I had taken an arrow in the knee.

Elder Scrolls & RPG - Skyrim NPCs & Inworld AI

Ai interacting to Ai

One paper that i found beyond fascinating was Interactive Simulacra of Human Behavior.



Generative Agents

arXiv, April, 2023,

Figure 3: A morning in the life of a generative agent, John Lin. John wakes up around 6 am and completes his morning routine, which includes brushing his teeth, taking a shower, and eating breakfast. He briefly catches up with his wife, Mei, and son, Eddy, before heading out to begin his workday.

support. What do you think?
John: I think it's great that he's running. I'm curious to see who else is running and how the election will turn out.

Gradually, Sam's candidacy becomes the talk of the town, with some supporting him and others remaining undecided.

They created an environment that allowed ml agents to communicate to one another. One of the most exciting outcomes was that one agent organised a birthday party and proceeded to invite other ml agents to the party.

arXiv, April, 2023, J.S. Park, J.C. O'Brien, C.J. Cai, M. Morris, P. Liang, M.S. Bernstein

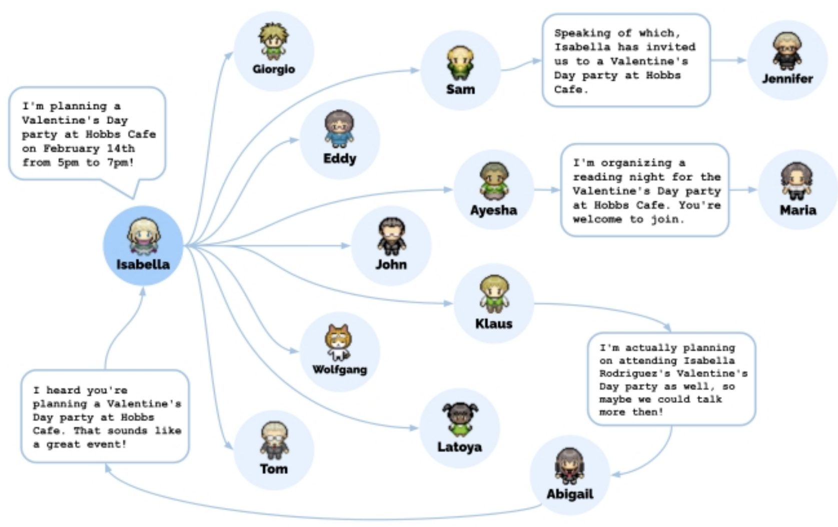
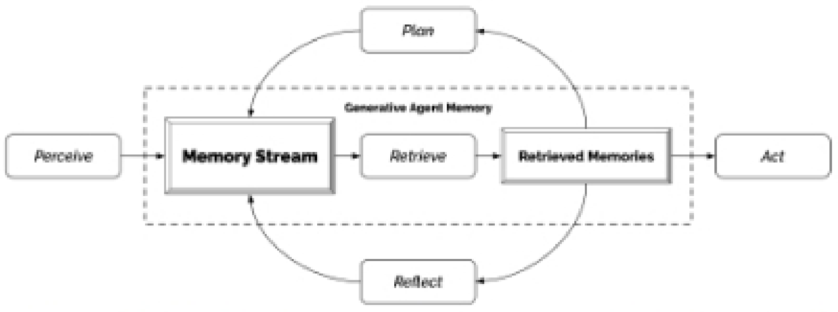

I'm planning a Valentine's Day party at Hobbs Cafe on February 14th from 5pm to 7pm!
Isabella
Giorgio
Sam
Speaking of which, Isabella has invited us to a Valentine's Day party at Hobbs Cafe.
Jennifer
Eddy
Ayesha
I'm organizing a reading night for the Valentine's Day party at Hobbs Cafe. You're welcome to join.
Maria
John
Klaus
Wolfgang
I'm actually planning on attending Isabella Rodriguez's Valentine's Day party as well, so maybe we could talk more then!
I heard you're planning a Valentine's Day party at Hobbs Cafe. That sounds like a great event!
Tom
Latoya
Abigail

Figure 9: The diffusion path for Isabella Rodriguez's Valentine's Day party. A total of 12 agents heard about the party at Hobbs Cafe by the end of the simulation.

The following goes more in detail over the implementatation design for one agent:

xiv, April, 2023, J.S. Park, J.C. O'Brien, C.J. Cai, M. Morris, P. Liang, M.S. Ber

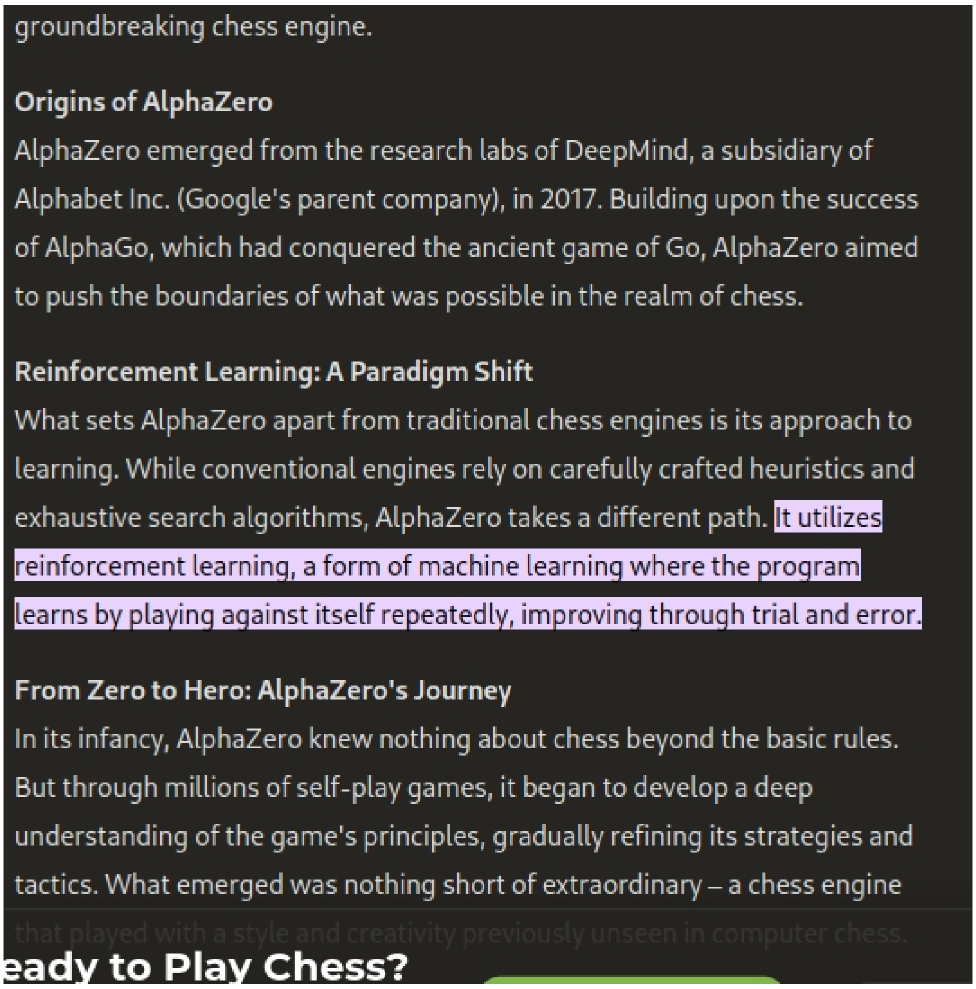


igure 5: Our generative agent architecture. Agents perceive their environment, and all perceptions are saved in a com ensive record of the agent's experiences called the memory stream. Based on their perceptions, the architecture retri levant memories, then uses those retrieved actions to determine an action. These retrieved memories are also used to f nger-term plans, and to create higher-level reflections, which are both entered into the memory stream for future use.

This design grants access to managing memories by long-time storage of relevant info.

Out-performing Human Performance

There are chess bots being developed that use machine learning in an attempt to "solve" the game of chess. So it is clear to say there is is a lot of incentivise towards acomodating machine learning algorithms into games.


groundbreaking chess engine.

Origins of AlphaZero
AlphaZero emerged from the research labs of DeepMind, a subsidiary of Alphabet Inc. (Google's parent company), in 2017. Building upon the success of AlphaGo, which had conquered the ancient game of Go, AlphaZero aimed to push the boundaries of what was possible in the realm of chess.

Reinforcement Learning: A Paradigm Shift
What sets AlphaZero apart from traditional chess engines is its approach to learning. While conventional engines rely on carefully crafted heuristics and exhaustive search algorithms, AlphaZero takes a different path. It utilizes reinforcement learning, a form of machine learning where the program learns by playing against itself repeatedly, improving through trial and error.

From Zero to Hero: AlphaZero's Journey
In its infancy, AlphaZero knew nothing about chess beyond the basic rules. But through millions of self-play games, it began to develop a deep understanding of the game's principles, gradually refining its strategies and tactics. What emerged was nothing short of extraordinary – a chess engine that played with a style and creativity previously unseen in computer chess.

Part II

Field Study

Field Study - Introduction

The Field Study chapters will present a brief explication of the moving parts that are involved in computer graphics rendering.

First chapter is the one with more focus on embedded-systems concepts Second chapter goes through graphics abstraction layers Third chapter presents an overview of the popular solutions offered to end-users.

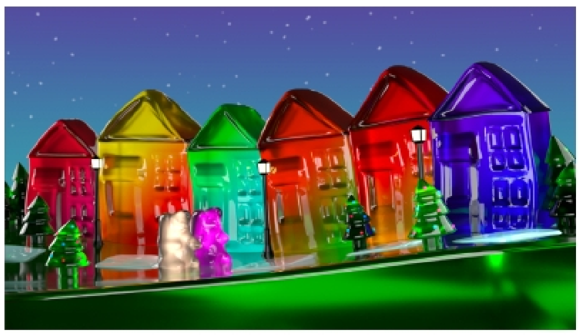
Real-World use cases

Student Projects - Stanford CS148 Computer Graphics and Imaging Course

Valerie Tang
CS148
13 December 2023

Note: Display discrepancies in Blender render and downloaded file

On my system, I am running into the issue where the downloaded image file is less saturated than the render on Blender itself, despite trying many different methods in office hours to remedy it. After cross-checking with several TAs, it seems that the amount of saturation in the image might depend on each person's display. I submitted the image file itself (the less saturated one), but I have attached my view of the Blender render here for reference as this is how I intended the image to look:

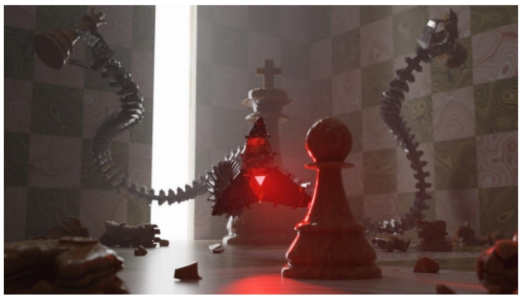


Project Requirements

i. I leveraged the power of ray tracing as the nature of my project is very "jello"-like, using a lot of transparent materials, which required transmission, reflection and good illumination. The light, shadow and color bleeding around the jello structures played a huge role in the aesthetic of the scene, and I would not be able to make it look as realistic without ray tracing.

The king in a chessboard has limited power. I wanted to reimagine the king as a malevolent ruler with disproportionate power that he does not mind using on his own side.

For this final image, a king with three robotic arms, inspired by Doctor Octopus from Spiderman, is holding a bishop and a queen. The scene is captured right before he tries to grab a pawn with its open claws. The ground is a chessboard, and the chessboard-patterned walls are also closing in. Some chess pieces are fragmented, implying that they were crushed by the king.



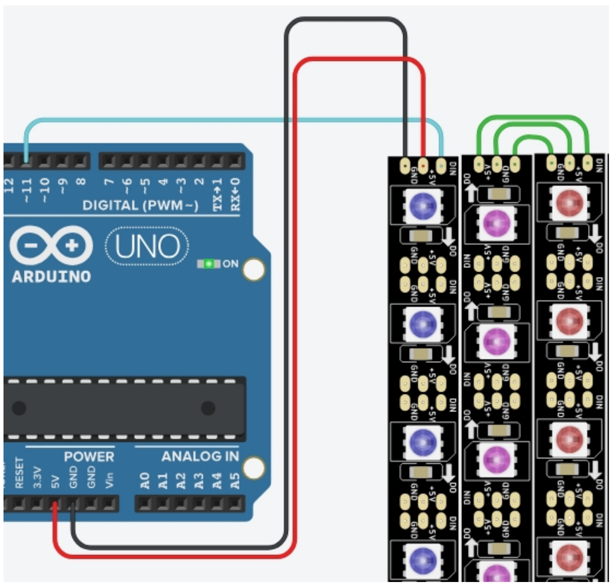
Inspiration board

Field Study - Hardware

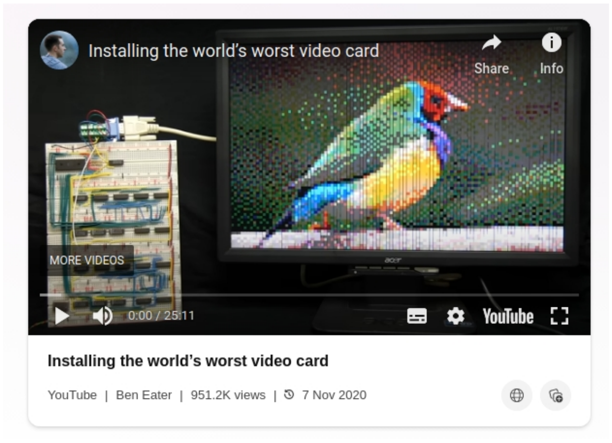
This chapter brings focus towards embedded systems by being a brief description of the process of getting the LEDs on our screens to display whatever the computer's video board decides to render.

The connection between screen and motherboard

Proof of concept using Arduino



Proof of concept using embedded circuits



Installing the world's worst video card

YouTube | Ben Eater | 951.2K views | 7 Nov 2020

Field Study - Software

This chapter accepts the embedded solutions as they are and develops solutions that step forward. The usual philosophy is developing abstraction layers.

OPENGL - Motivation

There are a few options when choosing a graphics abstraction solution. The most popular in the game development industry are Vulkan and DirectX. DirectX is more appropriate when it comes to Windows-specific optimizations, while Vulkan profits from Low-level control and performance and performs better at High-performance applications with multi-threading.

The only disadvantage to both those solution is that neither of them is as documented as opengl. Also, opengl is more popular in the educational/academic space and felt like the more appropriate choice.

OPENGL - Features

OpenGL's extensive documentation comes with both positives and negatives. Being one of the oldest solution to this problem it had seen multiple refactoring stages throught the years, this is best observed when realising there is a new revision of the opengl superbible released every couple of years. Each presenting the usual "How-to" projects and also acting as an update journal.

In the following i will briefly talk about popular opengl features.

PHIGS Standard Compliant

Offers primitive support and for their attributes by using Rendering Algorithms that use the low-level per pixel drawing functions extensively.

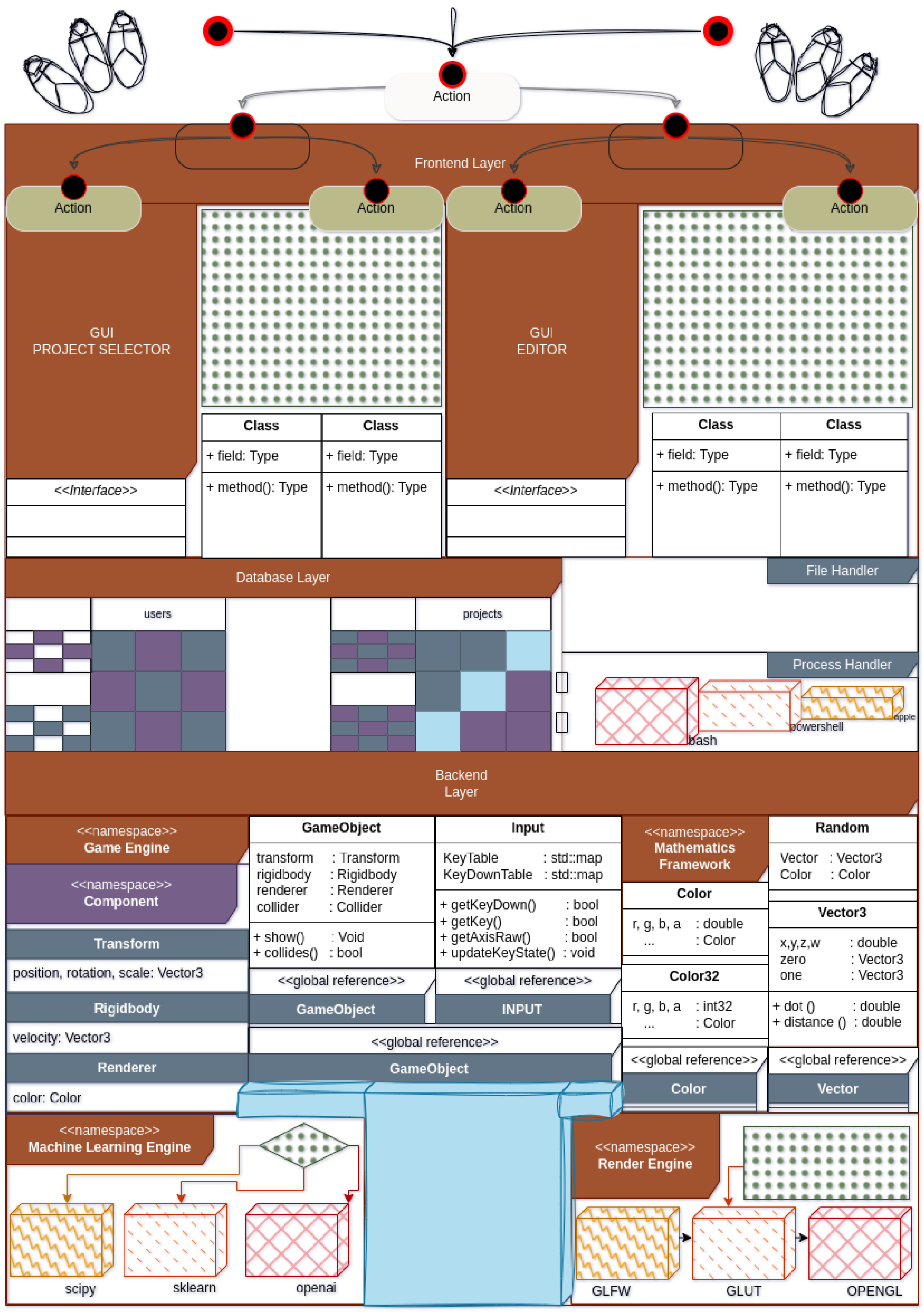
Part III

Implementation - Overview

DVD bouncer

Pong, The Game

Implementation - Architecture


Action
Frontend Layer
Action
Action
Action
Action
GUI
PROJECT SELECTOR
GUI
EDITOR
Class
Class
+ field: Type
+ field: Type
+ method(): Type
+ method(): Type
Class
Class
+ field: Type
+ field: Type
+ method(): Type
+ method(): Type
<<Interface>>
<<Interface>>
Database Layer
users
projects
File Handler
Process Handler
bash
powershell
apple
Backend
Layer
<<namespace>>
Game Engine
<<namespace>>
Component
Transform
position, rotation, scale: Vector3
Rigidbody
velocity: Vector3
Renderer
color: Color
GameObject
transform : Transform
rigidbody : Rigidbody
renderer : Renderer
collider : Collider
+ show() : Void
+ collides() : bool
<<global reference>>
GameObject
Input
KeyTable : std::map
KeyDownTable : std::map
+ getKeyDown() : bool
+ getKey() : bool
+ getAxisRaw() : bool
+ updateKeyState() : void
<<global reference>>
INPUT
<<global reference>>
GameObject
<<namespace>>
Mathematics
Framework
Color
r, g, b, a : double
... : Color
Color32
r, g, b, a : int32
... : Color
<<global reference>>
Color
Random
Vector : Vector3
Color : Color
Vector3
x,y,z,w : double
zero : Vector3
one : Vector3
+ dot () : double
+ distance () : double
<<global reference>>
Vector
<<namespace>>
Machine Learning Engine
scipy
sklearn
openai
<<namespace>>
Render Engine
GLFW
GLUT
OPENGL

I intent for this project to become a collection of technologies that can be interconnected. Similar to how Internet-of-Things solutions involve different running parts for achieving one goal, as such i want my software making environment to be as versatile as possible.

One obvious start impediment is going past the first couple of hello-world projects until you find a tool that does the job.

The percentage of availability of one component can be determined by the color of one's title.

bright colors means more relevant to this iteration.

neutral colors suggests the component's state of integrity and integration.

non-colors and you're looking at a component that hasn't been integrated yet.

Future expansion possibilities are in color.

- **FrontEnd (PyQt GUI programming interface)**

  integration dispatch layer

- Backend (Python Machine Learning Interface)

  communication socket

- **Backend (C/C++ Render Engine,Math Engine)**

The backend

There are two main languages used in this backend implementation:

in C/C++ we have: Rendering Engine, Vectorial MathEngine, GameEngine.

in Python we have: pyqt editor, Bash File Manager, Machine Learning Interface, Django.

The Vectorial Math Engine

Mathematical Framework with Homogenous Representation Support

Intuitively, for storing a 3D point one might think about using a vector of length 3 (maybe call them x, y, z) and have a great day. Well, this datastructure is good *enough* for most cases. There is, of course, one little edge-case in one of those cases where we might be needing a fancier solution. Also, this fancy solution somewhat simplifies the other calculations as well.

This new datastructure implies using a k+1 dimentional vector space for representing k dimensional entities.

Meaning that, in our application's purposes, a Vector3 class should store 4 elements. [More about this in future work]

```
class Vector3
{
  public:
    double x, y, z, w;
    Vector3(double _x, double _y = 0, double _z = 0, double _w = 1)
             : x(_x), y(_y), z(_z), w(_w) { }
};
```

$$\mathbf{v_1} = \begin{pmatrix} x_1 & y_1 & z_1 & w_1 \end{pmatrix}.$$

$$\mathbf{v_2} = \begin{pmatrix} x_2 & y_2 & z_2 & w_2 \end{pmatrix}.$$

Please do not be afraid of the w.

Since w is defined by a pretty straight-forward formula and the vector **usually** behaves like a point:

```
    Vector3 v = new Vector3::one * 7
    Debug::Log(v); // <7,7,7,1>
```

Data Structures

Vector Multiplication:

$$\mathbf{v_1} \cdot \mathbf{v_2} = (x_1 x_2) + (y_1 y_2) + (z_1 z_2) + (w_1 w_2) \tag{1}$$

Dot Product of Vectors:

The dot product $\mathbf{v}_1 \cdot \mathbf{v}_2$ between vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ is calculated as:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = v_{1,1} \cdot v_{2,1} + v_{1,2} \cdot v_{2,2} + \cdots + v_{1,m} \cdot v_{2,m}$$

Constructing a Matrix from Vectors:

Let $\mathbf{v}_1 = (v_{1,1}, v_{1,2}, \ldots, v_{1,m})$ be a vector representing the row elements.

Let $\mathbf{v}_2 = (v_{2,1}, v_{2,2}, \ldots, v_{2,n})$ be a vector representing the column elements.

```
  v1 = Random::Vector3();
  v2 = Random::Vector3();
```

The resulting matrix $\mathbf{M}$ formed by these vectors is:

$$\mathbf{M} = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,m} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,m} \end{pmatrix}$$

Matrix Multiplication (Traditional):

$$C = A \times B \quad \text{where} \quad C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj} \tag{2}$$

Strassen Matrix Multiplication (Recursive):

$$\mathbf{M}_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \tag{3}$$

$$\mathbf{M}_2 = (A_{21} + A_{22})B_{11} \tag{4}$$

$$\mathbf{M}_3 = A_{11}(B_{12} - B_{22}) \tag{5}$$

$$\mathbf{M}_4 = A_{22}(B_{21} - B_{11}) \tag{6}$$

$$\mathbf{M}_5 = (A_{11} + A_{12})B_{22} \tag{7}$$

$$\mathbf{M}_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \tag{8}$$

$$\mathbf{M}_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \tag{9}$$

Software Engineering

There are two main coordinate systems:

Cartesian Coordinates:

$$(x, y, z) \tag{10}$$

Polar Coordinates:

$$(r, \theta, \phi) \tag{11}$$

Transformation Formulas:

$$r = \sqrt{x^2 + y^2 + z^2} \tag{12}$$

$$\theta = \arctan\left(\frac{y}{x}\right) \tag{13}$$

$$\phi = \arccos\left(\frac{z}{r}\right) \tag{14}$$

In the following, we will take a look over the possible operation that are extensively used:

Operations in Computer Graphics

Translation:

$$T(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{15}$$

Scaling:

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{16}$$

These equasions must be really easy to use. For this i have chosen the option that highly resembles Unity's ecosystem.

```
void start();
void update();
int main() { Awake(); }
void Awake() {
  RenderEngine::setStart(start);
  RenderEngine::setUpdate(update);
  RenderEngine::setFixedUpdate(fixedUpdate);
  // MUST BE CALLED LAST
  RenderEngine::START(true);
}
void start() {
  Gameobject go; int speed = 0.01;
  Debug::Log(go.transform.name)
  Debug::Log(go.transform.position)

  go.transform.translate(Vector3::Right * speed)
  Debug::Log(go.transform.position)
  go.transform.position = Vector3::one * Math::sqrt(Math::pi);
  Debug::Log(go.transform.position)
}
```

The implementation feels self-explanatory from here. But for reference i advise source-code

Here lives the little inconvenience i mentioned at the beginning of the chapter that justified Homogenous Coordinates.

Rotation with Euler Angles (XYZ order):

$$R_{XYZ}(\alpha,\beta,\gamma) = R_X(\alpha)\cdot R_Y(\beta)\cdot R_Z(\gamma) \tag{17}$$

where

$$R_X(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{18}$$

$$R_Y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{19}$$

$$R_Z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{20}$$

Gimbal Lock Issue: Euler angles suffer from gimbal lock, where two of the three rotational axes align, leading to a loss of one degree of freedom.

Solution: Quaternions

$$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(u_x i + u_y j + u_z k) \tag{21}$$

where θ is the rotation angle and (u_x, u_y, u_z) is the unit vector representing the axis of rotation.

The opengl Rendering Framework

Color32

```
class color32
{
  public:
    unsigned int r, g, b, a;

    color32(double grayscale)
           : color(grayscale, grayscale, grayscale, 1.0f) { }
    color32(double _r, double _g, double _b, double _a = 1.0f) {   }
};
```

As you can see, similar to the vector class, the color datastructure is also a 4D vector. This will come in handy when dealing with shaders.

Opposite Colors

```
  class color32
  {
    public:
      unsigned int r, g, b, a;

      color32(double grayscale)
             : color(grayscale, grayscale, grayscale, 1.0f) { }
      color32(double _r, double _g, double _b, double _a = 1.0f) {   }
  };




```

```
void start();
void update();
int main() { Awake(); }
void Awake() {
  RenderEngine::setStart(start);
  RenderEngine::setUpdate(update);
  RenderEngine::setFixedUpdate(fixedUpdate);
  // MUST BE CALLED LAST
  RenderEngine::START(true);
}
void start() {
  Gameobject go;
  go.transform.position = Random::Vector3().normalised *  Random::Value
 (-5, 5);
  Debug::Log(go.transform.position)
}

```

Primitives

```

point(x, y, c); // Changes the color of the pixel at location <x, y> to c

line(<<x>, <y>>,
     <<x>, <y>>); // Draws a line between 2 points

background(color);

square(point1, point2);
fill(color);
circle(point , radius);

noStroke();
```

These functions are the fundamental building blocks for rendering graphics on a screen in OpenGL. For the existance of these functions, OpenGL implements various procedures to draw shapes and perform graphical operations.

Points and Lines

Points and lines are basic graphical primitives. A point is represented by its coordinates (x, y) on the screen, and a line is represented by a linear equation.

Line-Drawing Procedure

A line can be represented by the equation:

$$y = mx + b \tag{22}$$

where m is the slope of the line, calculated as:

$$m = \frac{\Delta y}{\Delta x} \tag{23}$$

The Machine Learning libraries interface

The Frontend

pyroGamer - The PyQt Editor

Introduction

The frontend component is called "the PyroGamer project" and is organized into several directories and files. It includes functionalities for the Editor, Hub, and Splash interfaces. These components are managed through various Python scripts and stylesheets. The overall structure can be understood through the following.

Directory Structure

$$\text{Directory } D := \{d_1, d_2, \ldots, d_n\}$$

$$\text{typeOf}(d_i) \in \{\text{File}, \text{Directory}\}$$

Components

Most of the sub-modules start as this:

$$F_m = S_m = \{\text{\_\_init\_\_.py}, \text{\_\_main\_\_.py}\}$$

And then each of them comes with their own required technologies.

$$\text{let hub } H \xrightarrow{\text{depend}} \text{cli interface} \wedge \text{FileManager} \wedge (\text{gui engine} \wedge \text{icons})$$

$$\text{let editor } E \xrightarrow{\text{depend}} \text{Hierarchy} \wedge \text{SceneView} \wedge \text{Inspector} \wedge (\text{Assets} \wedge \text{Terminal})$$

Configs

The Configs sub-directory contains scripts for configuration management:

$$C = \{\text{FileManager.py}, \text{\_\_init\_\_.py}, \text{\_\_main\_\_.py}\}$$

```python
import os

class FileManager:
    def __init__(self, path):
        self.path = path

    def list_files(self):
        return os.listdir(self.path)
```

Listing 1: FileManager.py

Elements

The Elements sub-directory is defined as:

$$L = \{\text{Pages.py}, \text{Tabs}\}$$

where Tabs itself is a set:

$$T = \{\text{Assets.py}, \text{Hierarchy.py}, \text{Inspector.py}, \text{SceneView.py}, \text{Terminal.py}\}$$

```python
class AssetsTab:
    def __init__(self):
        self.assets = []

    def add_asset(self, asset):
        self.assets.append(asset)
```

Listing 2: Assets.py

FileManager and SceneManager

Both FileManager and SceneManager sub-directories are represented as:

$$F_m = S_m = \{\text{\_\_init\_\_.py}, \text{\_\_main\_\_.py}\}$$

```python
if __name__ == '__main__':
    print("Initializing Manager")
```

Listing 3: $_{main.py}$

Styles

The Styles sub-directory contains CSS files for styling:

$$S_t = \{\text{createProjWindow.css}, \text{noProjectWindow.css}, \text{projectWindow.css}\}$$

Hub

The Hub component is represented by the set H:

$$H = \{\text{Configs}, \text{Elements}, \text{icons}, \text{\_\_main\_\_.py}\}$$

Configs

The Configs sub-directory in Hub contains:

$$C_h = \{\text{FileManager.py}, \text{\_\_main\_\_.py}\}$$

Elements

The Elements sub-directory in Hub is:

$$L_h = \{\text{Tables.py}, \text{Tabs.py}\}$$

Splash

The Splash component contains:

$$S_p = \{\text{images}, \text{\_\_main\_\_.py}\}$$

Inter-component Relationships

The interactions between different components can be represented using functions and mappings. Let $f : E \to H$ represent the function mapping elements from the Editor to the Hub. Similarly, $g : H \to S_p$ represents the mapping from Hub to Splash.

File Management

File management across different components is handled by the scripts in Configs and FileManager directories. Let $\mathcal{F}$ be the set of file management scripts:

$$\mathcal{F} = C \cup F_m \cup C_h$$

Styling

Styling is governed by CSS files in the Styles directory:

$$\mathcal{S} = S_t$$

The overall relationship between these parts can be expressed as:

$$\text{Frontend} = \mathcal{F} \cup \mathcal{S} \cup (L \cup L_h) \cup S_p$$

Conclusion

The frontend component of PyroGamer is a well-structured combination of directories and files. Each part has a specific role, and their interactions can be mathematically represented to understand the flow and dependencies within the system.

Part IV

Results

Formal Proof

In this chapter, we will inspect the criteria to which a piece of software needs to qualify in order to be considered part of the game engines set.

Natural Language Statement

Wikipedia ”A game engine is a software framework primarily designed for the development of video games and generally includes relevant engines.”

Let S be a piece of software.

Formal Definition

$$\text{isGameEngine}(S) = \begin{cases} \text{true}, & \text{if isFramework}(S) \wedge \text{hasEngines}(S) \\ \text{false}, & \text{otherwise} \end{cases}$$

$$\text{isSoftwareFramework}(S) = \begin{cases} \text{true}, & \text{if Generic}(S) \wedge \text{stdDeployed}(S) \\ \text{false}, & \text{otherwise} \end{cases}$$

Let GameEngines::PossibleComponents be the collection of all the possible engines in any game engine.

$$\text{GE::PC} = \begin{cases} \text{Rendering}, \text{Physics}, \text{Scripting}, \text{Sound}, \text{AI}, \text{Robotics}... \end{cases}$$

$$\text{hasEngineSuite}(P) = \begin{cases} \text{true}, & \text{if } (|S.\text{engines}| > 0) \wedge (S.\text{engines} \subseteq \text{GE::PC}) \\ \text{false}, & \text{otherwise} \end{cases}$$

Formal Proof

In this section, we will formally verify whether PyroGamer qualifies as a game engine based on the defined criteria.

Software Framework Verification

To determine if PyroGamer satisfies the software framework criterion:

$$\text{isSoftwareFramework(PyroGamer)} = \begin{cases} \text{true}, & \text{if Generic Functionality(PyroGamer)} \wedge \text{Standard} \\ \text{false}, & \text{otherwise} \end{cases}$$

Where:

- Generic Functionality(PyroGamer): PyroGamer provides generic functionality that can be adapted by user-written code, such as game logic scripts and customization options.
- Standard Deployment(PyroGamer): PyroGamer provides a standard way of building and deploying game applications, ensuring consistency and ease of use across different projects.

Suite of Relevant Engines Verification

To verify if PyroGamer includes a suite of relevant engines:

$$\text{hasEngineSuite(PyroGamer)} = \begin{cases} \text{true}, & \text{if } (|S.\text{engines}| > 0) \wedge (S.\text{engines} \subseteq \text{GameEngines::Possi} \\ \text{false}, & \text{otherwise} \end{cases}$$

Where:

- $|S.\text{engines}| > 0$: PyroGamer has a non-empty set of engines.
- $S.\text{engines} \subseteq \text{GameEngines::PossibleComponents}$: All engines in PyroGamer are part of the collection of possible engines in any game engine.

Conclusion

Based on the verification of both criteria:

- PyroGamer satisfies the software framework criterion by providing generic functionality and standard deployment features.
- PyroGamer includes a suite of relevant engines, covering rendering, physics, sound, scripting, AI, networking, streaming, memory management, threading, localization, scene graph, and video support.

Therefore, PyroGamer meets the formal definition of a game engine as defined by its software framework and suite of relevant engines.

Future Improvements

In this chapter, we outline future directions for the project and discuss upcoming enhancements and integrations.

Packaged Builds

The goal of this project is to eventually provide packaged builds for easier deployment and distribution. Specifically, we aim to package the project for AUR (Arch User Repository) and as a PyPi (Python Package Index) library.

Some progress has already been made towards this goal; an early build is available for installation using the following command:

```
pip install -i https://test.pypi.org/simple/ game-genie
```

Unfortunately, pip support had to be temporarily halted until the application reaches a more stable and mature state.

Performance Evaluation

In this section, we assess the performance aspects of the application, focusing on runtime efficiency and optimization opportunities.

Speed Analysis

To evaluate the speed of the application, various benchmarks and profiling techniques were employed. Preliminary tests indicate that the application performs adequately for its current scope. However, further optimization is planned to enhance performance in areas where execution time is critical.

Optimization Strategies

Several strategies for optimization have been identified, including algorithmic improvements, caching mechanisms, and parallel processing utilization. These strategies aim to reduce computational overhead and improve overall responsiveness.

Conclusion

The future improvements outlined here pave the way for a more robust and efficient application. By focusing on packaged builds and performance enhancements, we aim to provide users with a smoother and more optimized experience.

Chapter 1

Bibliography

- Author1, *Book1*, 2018
- Author2, *Boook2*, 2017