Suprannua Engine Architecture Document

1. Introduction

Suprannua Engine is a 2D platformer-oriented game framework for compiling simple, arcade-like PC games or visualisations for algorithms. The hallmark and namesake of this engine is a superannuated design where the visuals are minimally done with legacy OpenGL, and the architecture is structured as a collection of C functions.

This was a first time project that was designed beyond basic C programming exercises. Therefore, it was created with a limited knowledge of standard programming practices or even a solid plan of the architecture before having it implemented. It has since been refined to just get the components working as intended while being lenient on the coding style and original architecture (use of global variables, externs, etc).

2. Core

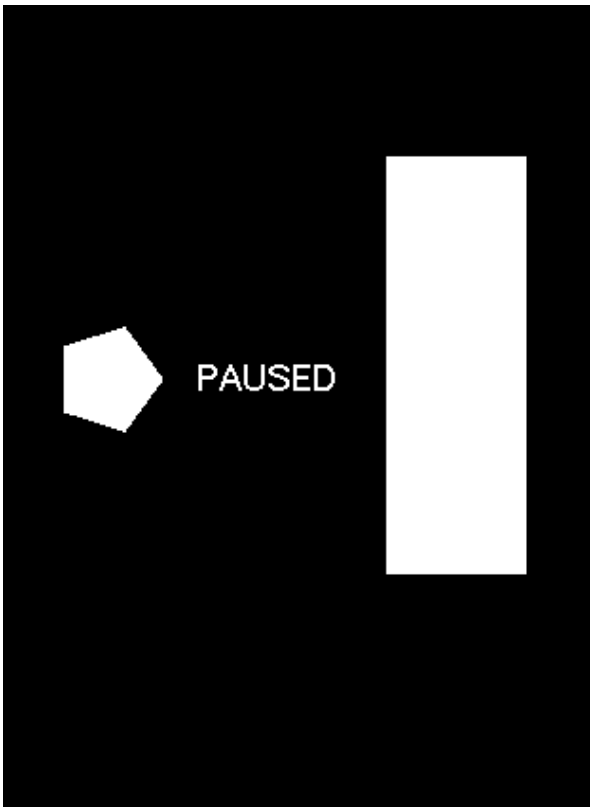
2.1 Data structures

Suprannua Engine's game objects are strictly Polygons, Blocks, Texts and one Camera. All game objects are instances of type defined structs. They all contain the Vertex struct to represent x and y positions against a world map. The world map and rectangular objects like the Camera and Blocks all contain the Rect struct to represent width and height. The world map x coordinate, from 0 to the maximum size, corresponds with the left to right. The y coordinate from 0 to maximum size corresponds with down to up.

```
typedef struct
{
    Property properties;
    Vertex vertices[MAX_POLYGON_SIDES];
    Vertex centre;
    double radius;
}RegularPolygon;

typedef struct
{
    Property properties;
    Vertex vertices[4];
    Vertex centre;
    Rect dimensions;
}Block;

typedef struct
{
    Vertex textPin;
    char textContent[128];
    unsigned char colour[4];
    unsigned char classification;
}Text;
```



Polygons and Blocks are the only objects that contain the Properties struct, which describes the object's;

- •Classification (Whether it is a background, foreground, entity, platform, etc)
- •Colour (Including alpha)
- •Edges (Geometric sides)
- •Angle (Only really applicable to Polygons)
- •BouncePercentage (Elasticity)
- •Mass
- •xVelocity
- •yVelocity

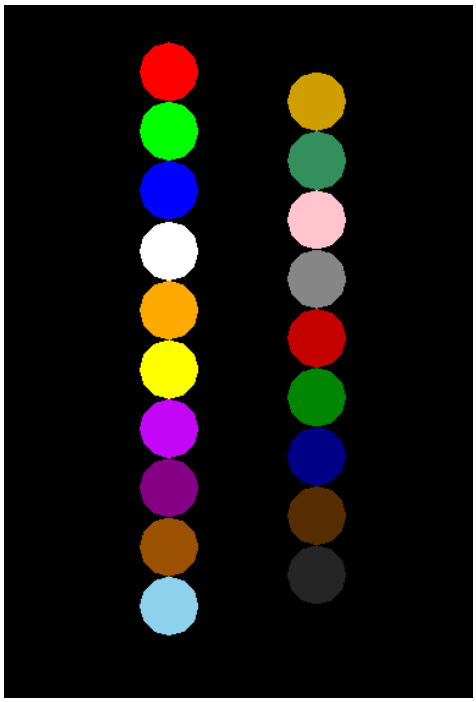
Text is different where in addition to position, it stores a string with colour and either an entity or heads up display classification (to pin to the world map or camera, respectively).

```
typedef struct
{
    unsigned char classification;
    unsigned char colour[4];
    int edges;
    double angle; //for rotation
    double bouncePercentage;
    double mass;
    double xVelocity;
    double yVelocity;
}Property;
```

2.2 Colour palette

There are 21 colours supported by the Suprannua Engine that range from black, white, rainbow colours and additional secondary colours with different shades. These values are stored in globally declared, hard coded arrays of unsigned chars:

```
unsigned char black[3] = { 0,0,0 };
unsigned char white[3] = { 255,255,255 };
unsigned char red[3] = { 255,0,0 };
unsigned char green[3] = { 0,255,0 };
unsigned char blue[3] = { 0,0,255 };
unsigned char orange[3] = { 255,165,0 };
unsigned char yellow[3] = { 255,255,0 };
unsigned char violet[3] = { 191,5,247 };
unsigned char purple[3] = { 128,0,128 };
unsigned char brown[3] = { 150,75,0 };
unsigned char skyBlue[3] = { 135,206,235 };
unsigned char gold[3] = { 204,153,0 };
unsigned char seaGreen[3] = { 46,139,87 };
unsigned char pink[3] = { 255,192,203 };
unsigned char grey[3] = { 128,128,128 };
unsigned char darkRed[3] = { 192,0,0 };
unsigned char darkGreen[3] = { 0,128,0 };
unsigned char darkBlue[3] = { 0,0,128 };
unsigned char darkBrown[3] = { 80,40,0 };
unsigned char magenta[3] = { 255,0,228 };
unsigned char darkGrey[3] = { 32,32,32 };
```



The alpha values for the game objects are defined by the editor and text modules of the engine. By default the object is given a full 255 value so that they appear opaque. In the coming chapters, changing alpha values and transparency will be explained.

2.3 States

The engine holds input states in an array of boolean values called "keyStates." These booleans are indexed by ASCII values to register whether certain keys are pressed. For example, if the "a" key is pressed, the engine will take the ASCII value of "a", which is 97 and store a true value in array cell 97.

```
bool keyStates[128] = { false };
```

Having an array of these input states allow the engine to register multiple inputs at once so the player may move an object while performing another action at the same time, like running and jumping.

The gameState variable is primarily used for the game input definitions. It is an unsigned char that signals whether the game is considered to be in "GAMEPLAY", "MENU", or perhaps other custom states programmed by the developer.

```
unsigned char gameState = GAMEPLAY;
```

For example, if the engine were in GAMEPLAY state, the WASD keys could be used for object

movement, but if it were in MENU state, the WASD keys could be used for moving the camera or selecting and scrolling through a list of texts.

There are also variables for stored objects and renderables. Nothing is dynamically allocated, so these variables are used to keep track of the workload size of compute and rendering loop iterations.

```
int storedBackgrounds = 0;
int storedPlatforms = 0;
int storedEntities = 0;
int storedForegrounds = 0;
```

```
int storedPolygons = 0;
int storedBlocks = 0;
int storedTexts = 0;
```

Finally, there are additional boolean variables for registering whether debug interfaces should be rendered to screen:

```
bool isEngineStatsEnabled = false;
bool isGridEnabled = false;
```

2.4 Constants

The engine retains constants for its name, and version number, and various other useful values like the value of pi to 32 places, max 8-bit colour value, target frame rate and the frame time in milliseconds.

```
 /*Define engine constants*/

#define SOFTWARE                "Suprannua Engine"
#define VERSION                 " 1.0.0 "

#define PI                      3.14159265358979323846264338327795
#define FULL                    255 //Colour level
#define FRAME_RATE              60.0
#define FRAME_TIME_MILLISECS    1000.0/FRAME_RATE
```

Other constants include the maximum amount of objects statically allocated for arrays, maximum polygon sides, texts and audio files to store.

```
#define MAX_DEFAULT_OBJECTS     1000
#define MAX_POLYGONS            MAX_DEFAULT_OBJECTS
#define MAX_POLYGON_SIDES       100
#define MAX_BLOCKS              MAX_DEFAULT_OBJECTS
#define MAX_TEXTS               1000
#define MAX_AUDIO_FILES         50
```

There are enumerations for the colour palette, objects, object types, object attributes, prepositions, control modes, spin directions, gravitation, platform scrolling directions and audio types to specify to major modules how operations on objects are to be performed.

2.5 Entry point

The Suprannua Engine at one point only used the WinAPI. This was to build the program with only a window and icon. Now by preprocessor directives, #ifdef _WIN32 compiles the WinAPI portion of code if it's running on Windows, and uses regular int main() on other platforms. Other platforms are supported provided that they have support for FreeGLUT and SDL.

```
#ifdef _WIN32

int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPTSTR lpCmdLine,
    int cmdShow) //Win32 GUI based Application.
{
    initSDLAudio();

    int screenWidthPixels;
    int screenHeightPixels;
    char executableName[68];
    int argc = NULL;
    char **argv = NULL;

    glutInit(&argc, argv);

    screenWidthPixels = glutGet(GLUT_SCREEN_WIDTH) * 0.750;
    screenHeightPixels = (screenWidthPixels * 0.563);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(screenWidthPixels, screenHeightPixels);
    glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH) - screenWidthPixels) / 2,
        ((glutGet(GLUT_SCREEN_HEIGHT) - screenHeightPixels) / 2) - 20);
    glutCreateWindow(gameTitle);

    /*Assigns the resource icon to the executable of the same name as the game title.*/
    HWND hwnd = FindWindow(NULL, (gameTitle));
    sprintf(executableName, "%s.exe", gameTitle);
    HANDLE icon = LoadImage(GetModuleHandle((executableName)), MAKEINTRESOURCE(IDI_ICON1), IMAGE_ICON, 32, 32, LR_COLOR);
    SendMessage(hwnd, (UINT)WM_SETICON, ICON_BIG, (LPARAM)icon);
```

```
#else

/*For non-Windows platforms compatible with SDL and freeGLUT*/

int main(int argc, char **argv)
{
    initSDLAudio();

    int screenWidthPixels;
    int screenHeightPixels;

    glutInit(&argc, argv);

    screenWidthPixels = glutGet(GLUT_SCREEN_WIDTH) * 0.750;
    screenHeightPixels = (screenWidthPixels * 0.563);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(screenWidthPixels, screenHeightPixels);
    glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH) - screenWidthPixels) / 2,
        ((glutGet(GLUT_SCREEN_HEIGHT) - screenHeightPixels) / 2) - 20);
    glutCreateWindow(gameTitle);
```

3. FreeGLUT API

4. Game Loop

5. Modules
5.1 2D Audio
5.2 2D Camera
5.3 2D Renderer
5.4 AI
5.5 Editor
5.6 Events
5.7 Geometry
5.8 Input
5.9 Physics
5.10 Text

6. Game
6.1 Game global variables

There are variables that are defined by a custom game entry point file. These define the title, dpad sensitivity for camera and movement, world size in metres, gravity from platforms, and gravity between polygons.

```
char gameTitle[64] = SOFTWARE VERSION" Standard Game Demo ";
Rect worldSizeMetres = { 200,50 }; // m
double dpadSensitivity = 10.0; // m/s
double cameraScrollSpeed = 50.0; // m/s
double platformGravity = 9.8; // m/s^2
double gravityConstant = 6.674E-11; // m/s^2
```