让 raylib 显示中文的尝试

raylib 支持 utf8 编码，但其默认字体只有最基本的 ASCII 字符，因此要想在 raylib 应用里显示中文就要加载一个能支持中文的字体

面临的问题

raylib 提供的几种字体加载方式都有缺陷：

1. `Font LoadFont(const char *fileName)`:一次性加载所有字符，消耗大量空间
2. `Font LoadFontFromMemory(...)`:可以选择性地加载字符，但是较为复杂

权衡之后我选择了第二种加载方式，因为这种方式虽然复杂一些，但是更加灵活。

动手解决

主要参考这位大佬的方法，但为了能支持动态加入文本进行了一些修改。

这种方式的复杂性从加载函数的函数原型就能看出来：

```
Font LoadFontFromMemory(const char *fileType, const unsigned char *fileData,
int dataSize, int fontSize, int *codepoints, int codepointCount);
```

这里解释下各个参数：

1. fileType:表示字体文件的类型，这里以.ttf 字体文件为例
2. fileData:字体文件在内存中的地址
3. dataSize:字体文件大小
4. fontSize:加载字体时的像素精度，这里以 200 为例
5. **codepoints**:字符的码点表示，注意它实际上是一个码点数组（称为**码点表**）
6. codepointCount:码点表的大小

这个函数的麻烦之处在于：

它的返回值只会包含传入的码点表对应字符的字体信息，所以每次出现新字符时都要更新字体。

基本思路如下：

1. 有一个 wstring 类型的字符串 allCharacters，保存着目前加载的所有字符
2. 在程序一开始就预载入一些常用字符和已记录字符
3. 每次绘制文字时，检查有没有遇到新的字符
4. 如果遇到了新字符添加到 allCharacters 中，并在检查完所有需要绘制的字符后更新字体
5. 利用最新的字体绘制文字内容
6. 在每一帧结束绘制之后清理旧字体
7. 更新记录字符

问题与可能的解决方案

从内存中获取字体的时间成本较高，而且每次遇到新字符时都需要重新从内存中重新获取新字体，当字符较多时游戏会出现明显卡顿。采用异步加载字体的方式可能会缓解。

相关代码

```
extern "C"{
  #include "raylib.h"
}
#include <string>
#include <list>
#include <locale>
using namespace std;
//字体路径
#define ASSET_FONT_PATH "./font.ttf"
//一些邪恶的全局变量，最好要封装到类里面
wstring allCharacters;
int fontFileSize;//字体文件的大小
unsigned char* fontFile;//字体文件
Font nowFont;//目前最新的字体
list<Font> usedFonts;//需要删除的旧字体
bool shouldClean=false;//清理旧字体标志

Font getFont(const wstring&);//更新并返回字体
void cleanFont();//清理旧字体
void DrawTextPlus(const wstring&,const Vector2&,const int,const Color&);//包装之后的绘制文字函数

int main(){
  InitWindow(1600,900,"Test");
  wstring textToDraw="这是一段测试文字";
  //用raylib提供的函数读取字体文件，fontFileSize是输出参数，得到文件的大小
  fontFile=LoadFileData(ASSET_FONT_PATH,&fontFileSize);
  //提前加载常用字，可以从文件读取，这里不另外写
  nowFont=getFont(PRELOAD_CHARACTERS);
  while(!WindowShouldClose()){
    BeginDrawing();
    ClearBackground();
    DrawTextPlus(textToDraw);
    EndDrawing();
    cleanFont();
  }
  //清理
  UnloadFileData(fontFile);
}

void DrawTextPlus(const wstring& text,const Vector2& pos,const int fontSize,const Color& color){
  //由于DrawTextEx()接受的文本参数是char*类型，wstring字符串要先转换成utf8型的string字符串
  wstring_convert<codecvt_utf8<wchar_t>> converter;
  string textInUTF8=converter.to_bytes(text);
  DrawTextEx(getFont(text),textInUTF8.c_str(),pos,fontSize,1,color);
}
Font getFont(const wstring& text){
```

```
  bool hasNewChar=false;
  for(const auto& c:text){
    if(allCharacters.find(c)==wstring::npos){
      allCharacters+=c;
      hasNewChar=true;
    }
  }
  if(hasNewChar){
    shouldClean=true;
    wstring_convert<codecvt_utf8<wchar_t>> converter;
    string allCharactersInUTF8=converter.to_bytes(allCharacters);
    int codepointCount=0;
    //使用raylib提供的工具函数来获得码点表，这里codepointCount是输出参数，得到的是字符数
量
    int*
codepoints=LoadCodepoints(allCharactersInUTF8.c_str(),&codepointCount);
    //把旧字体放进垃圾桶
    usedFonts.push_back(nowFont);
    //更新字体

nowFont=LoadFontFromMemory(".ttf",fontFile,fontFileSize,200,codepoints,codepointCount);
    UnloadCodePoints(codepoints);
  }
  return newFont;
}
void cleanFont(){
  for(auto& font:usedFonts){
    UnloadFont(font);
  }
  usedFonts.clear();
  shouldClean=false;
}
```