朋辈课程：C 语言小游戏设计实践

一、贪吃蛇小游戏规则

（1）由四面墙，水果，蛇构成

（2）蛇根据用户指令运动，撞到墙和自身则失败

（3）蛇需要通过吃水果来得分，蛇头到达水果位置则认为吃到了水果

（4）一旦水果被吃掉，则随机生成新的水果

思考：写出规则我们就迈出了第一步，然而，虽然这个规则符合我们的生活直觉，但它距离编程是不是还是有些远了？知道这些后，是不是仍然不知道如何写代码？

二、游戏架构分析

在这一步，我们继续细化内容，我们将用编程的思维重新审视游戏规则。对于所有的游戏对象，我们思考：如何存储？如何表示？它们的属性又该如何表示？

（一）游戏界面：由什么组成

游戏世界（包括墙壁）、水果、蛇

（二）用什么数据类型表示

游戏世界：可以直接打印，也可以保存到二维数组 grid 中，先处理再统一打印。墙壁其实就是数组的边界，我们通过字面常量 WIDTH、HEIGHT 来定义。定义完游戏世界的存储空间，我们接下来决定游戏的样式，我们规定，若一个网格为空区域，则 grid 相应位置的值为空格' '或点'.'，若为蛇的身体则值为'O'，水果则为'F'。

水果：我们定义变量 fruitX、fruitY 存储其坐标，显示时我们可以用字母 F（Fruit）显示。

蛇：蛇不能仅考虑为一条线，它其实是有方向的，从头到尾依次编号为 0 到 n-1。我们定义 snakeX[WIDTH * HEIGHT]、snakeY[WIDTH * HEIGHT]来存储蛇每个点的位置，按从头到尾的顺序。同时，我们需要知道蛇的长度才能知道数组目前的实际长度，因此再定义一个 length 变量。

思考：针对蛇的这种特性，是否能找到更适合的数据结构？（维护尾指针的循环链表）

操作：我们需要在游戏过程输入参数来决定游戏的运行，那么有哪些参数？首先显然是方向，蛇通过方向参数获知自己应该朝那边走，因此需要一个 direction 变量。吃水果是可以得分的，

因此又需要 score 变量。

```
#define WIDTH 20
#define HEIGHT 15
char grid[HEIGHT][WIDTH];
int snakeX[WIDTH * HEIGHT], snakeY[WIDTH * HEIGHT];  // 蛇身的 x 和 y 坐标
int length;          // 储存蛇的长度
int fruitX, fruitY;  // 储存水果的 x 和 y 坐标
int score;           // 游戏得分
int gameover;        // 游戏结束标志
char direction;      // 蛇当前的移动方向
```

（三）游戏的运行流程

一个游戏的运行流程（简化版）：（1）初始化：确定游戏进入后的初始状态，主要定义变量、开辟内存空间、对某些变量赋默认值等；（2）更新：这是一个无限循环。游戏会根据Δt、用户输入等更新所有游戏物体的状态，并依据这些状态将其显示出来。（3）退出并销毁：游戏过程中，游戏除了更新游戏物体状态外，也会不断检查游戏是否需要退出，若需要则退出循环，释放内存空间；

对于贪吃蛇游戏，我们首先需要初始化蛇的位置、长度和方向，这样游戏一开始我们就可以操作；我们也需要初始化游戏世界 grid（两个 for 循环初始化），才能直接显示画面；我们还需要初始化一个水果；然后还需要将分数初始化为 0。

（四）重新描述游戏规则

原则：所有对游戏对象的描述转换为对数据结构的描述；所有对操作、运动的描述转换对数据结构进行的处理步骤的描述（即算法，如蛇的长度 length 自增 1）；所有对状态、属性等的描述转换为特定数据类型（如用 int 型的 1 表示胜利，物理质量为 double 类型的 3.53）；实际不用写这么细，细化到能够一眼看出如何编程即可；

（1）游戏开始时，蛇为自定义长度，位于自定义初始位置，初始移动方向向左（任意方向都行），随机生成一个水果位置并显示，得分默认为 0；

（2）在每一个循环的时间片内，程序将检测玩家输入，玩家可以通过键盘输入'w'、's'、'a'和'd'，控制蛇分别向上、下、左、右移动。注意，蛇在移动过程中不能反向。如果玩家没有输入，就按照原方向移动并继续循环；

（3）在时间片内，蛇会根据当前设定的方向自动移动一格。蛇的位置更新是通过将蛇头的位

置移到下一个格子，然后清除蛇尾的位置来实现的；

（4）如果蛇头移动到的新位置与水果位置重合，蛇的长度会增加 1，玩家得分增加 10，并在屏幕上随机生成一个新的水果位置。新的水果位置不与蛇身体重合；

（5）根据目前的状态渲染新画面并显示得分。蛇的部分由'O'组成，水果显示为'F'；

（6）如果蛇头移动到的新位置是屏幕边界，或者与蛇身体的任何部分重合，那么游戏结束标志设置为 1，下一次循环就不会再进入；

（7）游戏会不断循环，直到游戏结束。每个循环周期中，游戏会获取用户输入，更新蛇的位置，检查是否吃到水果，然后重新绘制游戏界面，检查游戏是否结束。每一个周期加上 200 毫秒延迟时间，这决定了游戏的速度。

```
int main() {
    initialize();
    drawScene();
    while (!gameover) {
        getInput();
        updateSnakePosition();
        checkFruitEaten();
        drawScene();
        checkGameOver();
        Sleep(200);  // 控制游戏速度
    }
    return 0;
}
```

（五）游戏的渲染

在贪吃蛇游戏中，我们的帧率定为 5 帧/秒。有些游戏（如围棋）则只需要在落子时渲染新画面即可。动态帧率的控制也能节省一些性能。

思考：如果电脑性能有限，无法满足 5 帧/秒的需求，怎么办？

我们的游戏已经规定每 200 毫秒走一步，因此我们可以规定一个Δt 变量，记录新帧和旧帧的时间间隔（假设单位为秒），然后走(delta_t / 200)步

三、开始动手

即使看到这里，我相信大部分同学还是不会做。那么，当我么面对一个复杂的需求的时候，我们到底如何开始呢？一般的方案是，先实现一个最小的版本，仅包含几个简单内容。

（1）代码阅读。可以先阅读别人的代码。

（2）实现一个界面，假设蛇仅有一个点，仅实现点的运动和碰撞检测即可（见 01-最原始版本.c）。通过这个基础程序，我们可以体会游戏的渲染、输入的获取，界面的打印等；

```
int main() {
    initialize();
    while (1) {
        getInput();
        updatePointPosition();
        drawScene();
        Sleep(400);  // 控制速度
    }
    return 0;
}
```

（3）实现蛇的绘制，仅实现蛇的运动和碰撞检测。重点在于实现 updatePointPosition()，也就是根据输入的方向键更新蛇的坐标；

（4）加入水果、得分，进一步完善游戏

四、游戏设计常用函数扩展

（一）获取输入（无缓冲输入、跳过输入）

_kbhit()是从 conio.h 头文件中定义的函数，原型为 int_kbhit(void)。它的功能是实时检查键盘是否有输入，如果有，则返回一个非零值，否则返回 0。它不会等待用户按键，也不会从缓冲区中读取字符，也不会影响其他输入函数的执行。这个函数非常适合在游戏编程或者其他需要实时响应用户输入的场景中使用。例如，在一个实时战略游戏中，我们可以使用_kbhit()来检查用户是否按下了某个快捷键，如果按下，就立即执行相应的动作。

_getch()同样是从 conio.h 头文件中定义的函数，原型为 int_getch(void)。它的功能是从控制台读取一个字符，但不显示在屏幕上（不带回显）。它会等待用户按键，然后立即返回读取的字符的 ASCII 码，而无需等待回车键。如果读取失败，则返回 EOF。它会清除缓冲区中的第一个字符。这个函数常常用于接收用户的隐藏输入，例如在输入密码等敏感信息时，我们就可以使用_getch()来读取用户的输入，这样就可以防止旁人窥看到输入的内容。

_kbhit()和_getch()的配合使用：

```
if (_kbhit()) {
```

```
    char c = _getch();
}
```

_getche()与_getch()类似，但是它会将读取的字符回显到显示屏幕上（带回显）。它的原型也是 int_getche(void)。它会等待用户按键，然后立即返回读取的字符的 ASCII 码，而无需等待回车键。如果读取失败，则返回 EOF。_getche()常用于需要用户实时看到自己输入的情况，比如在开发命令行工具时，用户输入的每个字符都需要立即显示在屏幕上，这时就可以使用_getche()。

getchar()函数是从 stdio.h 头文件中定义的，原型为 intgetchar(void)。它的功能是从标准输入 stdin 获取一个字符（一个无符号字符）。它会等待用户按回车键，然后从缓冲区中读取第一个字符，并返回该字符的 ASCII 码。如果到达文件末尾或发生读错误，则返回 EOF。这个函数常用于实现基本的文本输入和输出。例如，在编写一个简单的文本编辑器时，我们可以使用 getchar()来读取用户的输入，然后处理并显示在屏幕上。

（二）清空输出

（1）方法一：system("cls")　　// stdlib.h

（2）方法二：ANSI 控制码

```
printf ("\033[2J");     // 清屏
printf ("\033[0;0H");   // 光标移动到左上角
```

（三）ANSI 控制码

https://zh.wikipedia.org/wiki/ANSI%E8%BD%AC%E4%B9%89%E5%BA%8F%E5%88%97

C 语言可以通过发送 ANSI 转义序列到命令行终端来控制命令行窗口。ANSI 转义序列是一种在文本模式下控制光标位置、颜色以及其他选项的方式，它们由 ANSI 标准定义，因此在各种终端和系统上都有一定的通用性。

ANSI 转义序列以 ESC 字符（ASCII 码为 27）开头，后跟一个或多个表示特定功能的字符。例如，要将文本颜色设置为红色，可以使用以下转义序列：printf("\x1b[31m");

这里，\x1b 是 ESC 字符的十六进制表示，[31m 是设置文本颜色为红色的 ANSI 代码。

ANSI 转义序列中的一些常见功能包括：

（1）光标位置：\x1b[row;colH 或\x1b[row;colf，将光标移动到给定的行和列位置（row 和 col）。

（2）清除屏幕：\x1b[2J，清除屏幕并将光标移动到左上角（home 位置）。

（3）清除行：\x1b[K，清除从光标位置到行尾的部分。

（4）设置颜色：\x1b[31m，设置文本颜色为红色。其他颜色可以通过替换 31 为其他数字来设置，如 32 为绿色，33 为黄色等。

（四）当堂小测试

（1）当堂作业：打印棋盘。打印一个 19*19 的围棋棋盘，尽量工整、美观，最好加上坐标；

（2）当堂作业：无缓冲输入 demo。程序很简单，就是“输出刚刚输入的字符”，但使用无缓冲输入；

（3）当堂作业：能实时显示系统时间的程序（需要以秒为单位实时更新）

（4）对于（3）中程序，可以尝试加上背景色和文字颜色；

五、贪吃蛇核心函数原理

（一）方向的获取

```
char direction;
void getInput() {
    char nextDirection;
    while (_kbhit()) {  // 检测是否有键被按下，并清空多余的输入
        nextDirection = _getch();  // 获取按下的键值
    }
    updateDirection(nextDirection);
    switch (nextDirection) {
        case 'w':
            if (direction != 'd') direction = 'u';  // 上
            break;
        case 's':
            if (direction != 'u') direction = 'd';  // 下
            break;
        case 'a':
            if (direction != 'r') direction = 'l';  // 左
            break;
        case 'd':
            if (direction != 'l') direction = 'r';  // 右
```

```
            break;
        }
    }
```

重点：_kbhit()和_getch()的配合使用。在其他游戏中可能需要等待用户输入，此时就不应使用_kbhit()。

思考：为什么获取 nextDirection 要用 while 循环？

（二）蛇坐标的更新

输入参数：这个算法需要什么？

首先，肯定需要蛇的坐标和蛇的长度；其次，需要方向，这样才知道往那边走；再然后需要知道游戏世界的范围（网格的数量）以免出界；

原则：在蛇坐标的最前方插入一个新的点作为头，同时将最后一个坐标删去。（画图演示）思考，这如何实现，有没有更合适的数据结构？

那么，蛇吃到水果后呢？蛇的长度要加一，那蛇的尾巴怎么办？我们的处理方法是，尾巴保持原来位置不变（不用删除），在开头按方向新增一个代表头的坐标即可。（画图演示）

```
void updateSnakePosition() {
    int prevX = snakeX[0];
    int prevY = snakeY[0];
    int prev2X, prev2Y;
    snakeX[0] = snakeX[0] + (direction == 'r') - (direction == 'l');  // 根据方向移动蛇头的 x 坐标
    snakeY[0] = snakeY[0] + (direction == 'd') - (direction == 'u');  // 根据方向移动蛇头的 y 坐标
    for(int i = 1; i < length ; i++) {
        prev2X = snakeX[i];
        prev2Y = snakeY[i];
        snakeX[i] = prevX;
        snakeY[i] = prevY;
        prevX = prev2X;
        prevY = prev2Y;
    }
    grid[snakeY[0]][snakeX[0]] = 'O';  // 新的蛇头位置
    grid[prevY][prevX] = '.';  // 清除蛇尾位置
}
```

（三）显示更新后的蛇

只需要将新的头改为0，将原来的蛇尾改成.即可。

（四）判定输赢

（1）判定蛇头是否超过边界

（2）将蛇头与蛇身逐个比较，看是否撞到自己

六、作业

（1）填充（1）部分，实现第一步（参见“01-最简单版本.c”）（尝试当堂完成）

（2）填充（2）部分，实现第二步（参见“02-加入蛇的版本.c”）

（3）自行实现第三步，做出最终游戏（“03-完整版本.c”，为空，需自己实现）

（4）完善、改进：加入游戏开始菜单和结束菜单，结束后用户可选择再来一盘或退出

（5）增加得分排行榜功能，用户玩游戏前先输入用户名，然后该局的得分就记录在在用户下。用户在游戏菜单中可以选择查看排行榜，查看自己的最高分，所有用户的得分排名等。（提示：通过文件长期记录信息）

（6）增加双人模式，两条蛇，多个水果，两人用不同的键操作，比拼谁的得分高，规则同样是不允许相撞；

（7）独立实现围棋小游戏，可落子即可（简单）（参见“围棋小游戏.c”，尝试当堂完成，并阅读学习代码）

（8）实现井字棋游戏，可落子，可判断胜负，有简单提示（中等难度，适合尝试自己完成）

（9）将围棋游戏改成五子棋游戏，可判断胜负，独立实现（较难，可参考围棋小游戏代码）

七、游戏引擎简介

下面的内容更多的是了解，当成阅读材料即可。

（一）游戏引擎总论

游戏引擎是一种软件系统，它为游戏开发者提供了一套通用的功能和工具，以便快速地创建和运行不同类型和风格的游戏。它的主要目标是抽象出游戏开发中的共通部分，使开发者能够专注于游戏的核心玩法和特色设计，而不是底层的技术细节。

（1）渲染引擎。渲染引擎负责将游戏中的元素（如三维模型、贴图、光照和阴影）转换为二维图像，并将其显示在屏幕上。它可以处理复杂的渲染技术，如光线追踪、环境光遮蔽、粒子效果等，

以创建高质量的游戏图像。

（2）物理引擎。物理引擎用于模拟游戏中的物理现象，如重力、碰撞、弹力和摩擦力等。这些模拟为游戏世界增加了真实感和趣味性，让玩家可以预测和利用物理规则来解决问题和挑战。

（3）动画引擎。动画引擎控制游戏中的角色和物体的动作，如行走、跳跃、攻击、受伤等。它可以处理复杂的动画技术，如骨骼动画、面部动画、布料模拟等，以创建生动、富有表现力的游戏角色。

（4）音效引擎。音效引擎负责处理游戏中的声音，如背景音乐、音效、对话等。它可以处理声音的混合、空间化、压缩等技术，以创建丰富、沉浸式的游戏氛围。

（5）脚本引擎。脚本引擎负责执行游戏中的逻辑，如游戏规则、故事情节、任务系统、人工智能等。它支持各种脚本语言，如 Lua、Python、C#等，让开发者可以用更简洁、更高层的方式来编写游戏逻辑。

（6）编辑器。编辑器提供了一个可视化的界面，让游戏开发者可以方便地设计和编辑游戏中的场景、角色、物体、事件等。它通常包括地形编辑器、材质编辑器、动画编辑器、粒子系统编辑器、UI 编辑器等，以满足各种开发需求。

（二）从硬件到游戏引擎的各个层次

（1）硬件层：此层包含基础硬件设备，例如 CPU、GPU、内存、硬盘、输入设备（键盘、鼠标）和输出设备（显示器、扬声器）。它提供底层硬件资源，如计算、存储和用户交互能力。

（2）驱动程序层：此层是操作系统和硬件设备之间的接口。它使用硬件层提供的设备资源，封装硬件指令，将具体的硬件操作转化为标准接口，供操作系统层使用。

（3）操作系统层：此层利用驱动程序层提供的设备接口，实现对硬件的抽象化管理。它封装了文件系统、进程管理、内存管理和设备驱动等基本服务，为上层图形 API 提供硬件设备的抽象接口和系统级服务。

（4）图形 API 层：此层基于操作系统层提供的抽象接口，封装了对 GPU 的高级编程接口，如 OpenGL、DirectX。它提供了创建 3D 图形和动画的能力，为上层的渲染引擎提供图形编程接口。

（5）渲染引擎层：此层使用图形 API 层提供的接口，实现了更高级的图形功能，如场景图、阴影、光照、后处理效果等。它为上层的游戏引擎提供更具体的图形渲染服务。

（6）游戏引擎层：此层集成了渲染引擎以及其他模块（如音频引擎、物理引擎、动画系统等）。它提供了创建游戏所需的各种工具和功能，包括但不限于图形渲染、音效处理、物理模拟、角色动作等，为上层的游戏开发提供全套解决方案。

（7）游戏层：这是最高层，由游戏开发者创建。它利用游戏引擎层提供的各种工具和功能，创建具体的游戏，实现游戏的各种玩法和特色设计，包括游戏逻辑、角色设计、关卡布局等。

（三）Start()和 Update()函数

在大多数游戏引擎中，Start()和 Update()是两个非常核心的函数，它们在游戏对象的生命周期中扮演着重要的角色。

（1）Start() 函数

Start()函数通常在游戏对象被初始化后立即调用一次，用于设置初始状态和参数。这可能包括初始化变量，设置对象属性，或连接到其他游戏对象等。在这个函数中执行的代码只会在对象创建时运行一次。

例如，如果你有一个游戏角色，你可能会在 Start()函数中设置它的初始位置，初始健康值，或者加载角色的模型和纹理等。

```
void Start()
{
    // Set the initial position of the character
    this.transform.position = new Vector3(0, 0, 0);
    // Set the initial health of the character
    health = 100;
    // Load the character model and texture
    LoadModelAndTexture();
}
```

（2）Update() 函数

Update()函数在游戏运行期间每帧都会被调用，用于处理游戏对象的持续更新，如移动，旋转，接收输入，进行计算等。这个函数是游戏逻辑的主要部分，你会在这里编写大部分的游戏代码。

例如，你可能会在 Update()函数中检查玩家的输入，更新角色的位置，检查角色的健康值，或者处理角色的死亡和复活等。

```
void Update()
{
```

```
    // Check the player's input
    CheckPlayerInput();
    // Update the character's position
    MoveCharacter();
    // Check the character's health
    if (health <= 0)
    {
        // Handle the character's death
        HandleDeath();
    }
}
```

八、物理引擎

（一）简介

在游戏开发中，物理引擎是其中的一个核心组件。它负责在游戏环境中模拟和处理各种物理现象，以创造出一个真实和生动的虚拟世界。从模拟刚体和软体的动态行为，处理物体间的碰撞检测与响应，再到复杂的流体动力学模拟，物理引擎都扮演着重要的角色。此外，物理引擎还需要解决诸如约束求解、物理材质管理、碰撞过滤以及空间划分等问题，以确保游戏的流畅运行和物理效果的真实性。

思考：有参数 v（当前瞬时速度向量），a（当前瞬时加速度向量），t（当前时间），s（当前的坐标），如何确定(t + delta_t)时刻的位置？

（二）实践

（1）实现动画，演示匀速直线运动，需要接收用户输入的速度值（当堂完成）

（2）用 x,y,z,r 四个分量描述三维空间的一个球，实现程序实现球的碰撞检测（当堂完成）

（3）实现动画演示平抛运动，假设触底后 $v_y := -0.7*v_y$，$v_x := v_x$，考虑帧的时间间隔Δt

九、扩展：用 C 语言实现简单图像处理

见“简单图像处理”文件夹。自行实现图像水平翻转、图像旋转九十度、用 Gamma 曲线调亮度的函数，并尝试运行。自行查看处理结果。PGM 格式一般的图像查看器可以打开，如果打不开可以使用网站：https://imagetostl.com/cn/view-pgm-online