CS100 HW8 样例游戏参考

去 年 作 业

注 意

- 请为本次作业分配充足的时间（参考：至少 3 天），并且不要到临近截止日期时才开始！

- 不要被本说明的长度吓到。它之所以长达 20 多页，是因为在每个游戏对象的细节说明里，我们有意做了很多不必要的文字重复。我们希望你能够从这些重复的内容里提取出共同点与区分点，通过自己的构思与设计来解决继承结构等问题，而不是被我们用我们设计的结构束缚住。

- 本文档只是样例实现的介绍，你不必全部实现，也不必完全遵循其中要求。你的目标是做出「你自己」的游戏，而不是「与样例一致」的游戏。本文档只提供作为参考的大量的实现流程与数值等细节。

去 年 作 业

- 如果对任何部分有疑问，在你提问之前，请先在本说明和 FAQ 中寻找答案，或试着游玩样例游戏并观察样例游戏的处理方式。

- 请尽量在 Piazza 上公开提问而非私下询问 TA。有相同疑问的其他同学也可以看到你的问题，你也能看到我们对更多问题的回答。这会更高效地解决你的疑问。

- 请一定经常保存备份（无论离线或在线）！如果你为了添加某一点功能使得整个程序崩溃或是在修复时越修越糟，简单地回退一版并比较不同，就可以快速解决问题。

◆ 好的，所以这么多东西我怎么写？

可想而知，从零开始写出一个完整的游戏绝不是能一蹴而就的小工程。本游戏的大量内容需要逐步完成。因此，我们将推荐先实现的部分以蓝色字体标出。我们建议你先阅读完整的题目说明，对这个游戏基本了解，之后再开工。写码时建议从蓝色字体部分开始逐步进行。附件“建议的开始流程.pdf”提供了一个清晰的前几步流程，相信你完成这几步之后就会驾轻就熟。

在在提供的文件之中，你不需要也不应当修改 `src/Framework` 路径下的文件。对于 `utils.hpp`，在图片素材加载错误时你可能需要修改其中 `ASSET_DIR` 字符串，并且你可以在其中定义新的项目来添加属于你自己的“杂交”内容（详见 FAQ：怎么添加新内容）。

下面是对游戏中每个组成部分的介绍。其中以符号开头的列表项是无关顺序的，数字或字母开头的列表项表示需要按顺序执行。

◆ GameWorld

◆ GameWorld::Init()

你的 `GameWorld::Init()`函数应当：

- 初始化任何用于记录关卡数据的成员变量。例如，初始波数为 0，阳光为 50。
- 为需要显示的关卡数据创建文本显示。需要使用 `TextBase` 类，使用方法与你创建的所有 `GameObject` 均类似，详细用法参见 FAQ。
- 创建背景。
- 创建 45 个种植位，`GAME_ROWS(5)`行 `GAME_COLS(9)`列，如右图：

  - 首个（左下角）位于`(x = FIRST_COL_CENTER, y = FIRST_ROW_CENTER)`。
  - 种植位之间的横向间距为 `LAWN_GRID_WIDTH`，纵向间距为 `LAWN_GRID_HEIGHT`。
- 生成位于顶端的植物种子按钮：
  - 首个按钮为向日葵种子，位于`(x = 130, y = WINDOW_HEIGHT - 44)`。
  - 其余按钮按顺序为豌豆射手、坚果墙、樱桃炸弹、双发射手，横向间距为 60。
- 生成铲子按钮，位于`(x = 600, y = WINDOW_HEIGHT - 40)`。

◆ GameWorld::Update()

你的 `GameWorld::Update()`函数应当：

1. 为游戏掉落新的阳光。掉落第一个阳光的时间为游戏开始后的第 180 个 tick（6 秒）。之后，每 300 个 tick（10 秒）掉落一个阳光。生成的阳光：

   - x 坐标为[75, WINDOW_WIDTH - 75]区间内的一个随机 int。
     - 我们已在 utils.hpp 中提供了生成随机闭区间 int 的函数 randInt()。
   - y 坐标为 WINDOW_HEIGHT - 1。
   - 将开始下落。

2. 为游戏生成新的僵尸。僵尸以波的形式出现，第一波僵尸将在游戏开始后的第 1200 个 tick（40 秒）生成，之后每波僵尸的生成间隔将加快。具体策略如下：

   - 当前生成的僵尸波数记为 wave。（游戏初始 wave 为 0 而首波僵尸为 wave 1）
   - 随机生成$\left\lfloor\frac{(15+wave)}{10}\right\rfloor$个僵尸。
   - 下一波僵尸将在 max(150, 600 - 20 * wave)个 tick 后生成。
     - 简单的验算：第 23 波将在第 22 波后 160 个 tick 生成，而之后每波的间隔固定为 150 个 tick。

3. 若在上一步中确定生成僵尸，则需要进一步随机决定每个僵尸的种类，对于每个生成的僵尸，策略如下：

   - 令 P1 = 20,
     P2 = 2 * max(wave - 8, 0),
     P3 = 3 * max(wave - 15, 0)，则：
   - 有 P1 / (P1 + P2 + P3)的概率，生成的僵尸为普通僵尸。
   - 有 P2 / (P1 + P2 + P3)的概率，生成的僵尸为撑杆跳僵尸。
   - 剩余的 P3 / (P1 + P2 + P3)的概率，生成的僵尸为铁桶僵尸。
   - 生成的僵尸 x 坐标为[WINDOW_WIDTH - 40, WINDOW_WIDTH - 1]区间内的一个随机 int，y 坐标为随机一行的 y 坐标（首行为 FIRST_ROW_CENTER，纵向间距为 LAWN_GRID_HEIGHT）。
     - 当你心里在抱怨“计算好麻烦”的时候，你又在骂数值策划了。果然，无论在什么游戏里，策划都会挨骂……

4. 遍历所有游戏对象(GameObject)，并依次调用它们的 Update()函数。

5. 检测碰撞。注意不要让同一碰撞被触发多次。可能发生的碰撞有：

   - 豌豆击中僵尸。
   - 爆炸影响到僵尸。

- 僵尸啃咬植物。
  - 撑杆跳僵尸的特殊碰撞检测会在其 Update()内完成。

6. 再次遍历所有游戏对象，将需要删除的对象从你的存储容器中移除。
   - 需要删除的对象应被标记为“死亡”或“HP 为零”等状态，GameWorld 才能找到。
7. 判断是否失败。若有任何僵尸的 x 坐标 < 0，则返回 LevelStatus::LOSING。
   - 在失败的显示画面上，我们预留了一个显示你完成的游戏波数的空位。你可以在此时创建一个文本来显示出你的游戏结果。创建文本请参考 FAQ。
8. 额外检测一遍僵尸是否在与植物发生碰撞。若某个僵尸没有和植物发生碰撞而在啃食植物，则使其正常走动。
   - 这种情况的发生是由于多个僵尸在某一帧啃死了同一棵植物。先啃的僵尸并不知道植物会死亡，只能保持啃食状态，所以需要额外判断一次以使其正常移动。
9. 更新你在游戏中创建的文本显示(TextBase)的内容，以正确显示需要的信息，例如阳光数、波数等。
10. 返回 LevelStatus::ONGOING，表示当前关卡在正常运行。

- **GameWorld::CleanUp()**

  你的 GameWorld::CleanUp()函数必须清空你所使用的容器。若你保存了其他对象，也需要将它们正确清除。

背景(Background)

- **当被创建时**
  - 背景的贴图编号为 IMGID_BACKGROUND。
  - 背景的位置为(x = WINDOW_WIDTH / 2, y = WINDOW_HEIGHT / 2)。
  - 背景的所在层级为 LAYER_BACKGROUND。
  - 背景的宽度为 WINDOW_WIDTH，高度为 WINDOW_HEIGHT。
  - 背景不具有动画，即，动画编号为 ANIMID_NO_ANIMATION。
- **当 Update()时**

  背景什么也不需要做。
- **当被点击时**

  背景什么也不需要做。

◆ 阳光(Sun)

◆ 当被创建时

- 阳光的贴图编号为 IMGID_SUN。
- 阳光的起始位置并不固定，而由创建它的代码决定。(因此，它的构造函数中必须包含此部分。)
- 阳光的所在层级为 LAYER_SUN。
- 阳光的宽度为 80，高度为 80。
- 阳光具有 ANIMID_IDLE_ANIM 动画。
- 阳光具有掉落时间。
  - ◆ 从天上掉落的阳光的掉落时间为[63, 263]之间的随机 int。这个掉落时间保证了它将掉在草坪上。
  - ◆ 向日葵产生的阳光将以抛物线落地，掉落时间为 12。

◆ 当 Update()时

1. 若阳光还未落地，则：
   - ◆ 如果它是从天上掉落的阳光，则它每刻向下移动 2 像素。
   - ◆ 如果它是向日葵产生的阳光，则它每刻向左移动 1 像素，并在竖直方向上做竖直上抛运动：其初速度（第一帧）为向上的 4 像素/刻，加速度为-1（方向向下）。
     - 这样的阳光运动轨迹是斜向左上的抛物线。在第 5 个 tick，它的 y 方向速度为 0，位于抛物线顶端。
2. 若阳光已经落地，它将不再移动，并在落地后 300 个 tick（10 秒）消失。

◆ 当被点击时

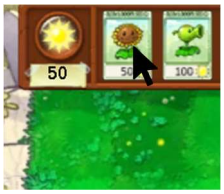
被点击时，阳光需要消失，并通知 GameWorld 获得 25 点阳光。

- 思考一下：要做到这点，阳光需要知道自己所在的 GameWorld，也就意味着需要储存这样一个成员。这个成员的类型应当是什么？GameWorld 创建阳光时，怎样把“所在的 GameWorld”这一信息告诉它？除了阳光，其他的类也可能需要知道所在的世界，那么这个信息应当存在哪儿？

◆ 种植位(Planting Spot)

种植位是位于草坪上的“隐形的兔子”。点击它可以把手上的植物种在它的位置。

◆ **当被创建时**



- 种植位的贴图编号为 IMGID_NONE，没有贴图。
- 种植位的位置并不固定，而由创建它的代码决定。（因此，它的构造函数中必须包含此部分。）
- 种植位的所在层级为 LAYER_UI。
- 种植位的宽度为 60，高度为 80。略小于草坪格，与植物大小一致。
- 种植位不具有动画，即，动画编号为 ANIMID_NO_ANIMATION。

◆ **当 Update()时**

种植位什么也不需要做。

◆ **当被点击时**

若玩家正拿着一个还未种下的种子，则“种下”它，即在种植位的位置生成一个对应的植物。

- 种下的植物由于位于更靠前的层级，将覆盖种植位。也就是说，有植物的种植位将不会再被点击到。
- 思考一下：既然在点击时才需要生成植物，那么玩家拿在手里的就并不是植物，当然也不是种子包（指左上角的按钮），毕竟手中的这个东西并没有贴图。而且，种植位需要根据“手里的东西”决定种下的是什么植物，说明这一信息存储在“手里的东西”上。这个东西究竟是什么？提示：另一种隐形的兔子！

◆ 向日葵种子(Sunflower Seed)

◆ **当被创建时**

- 向日葵种子的贴图编号为 IMGID_SEED_SUNFLOWER。
- 向日葵种子位于第一个种子包位置(x = 130, y = WINDOW_HEIGHT - 44)。
- 向日葵种子的所在层级为 LAYER_UI。
- 向日葵种子的宽度为 50，高度为 70。
- 向日葵种子不具有动画。
- 向日葵种子的价格是 50 阳光。
- 向日葵种子的冷却时间是 240 个 tick（8 秒）。

◆ **当 Update()时**

向日葵种子的 Update()并没有必须要做的事情，然而，基于你的实现，你也可以在 Update()中做些什么……

- **当被点击时**
  1. 如果玩家正拿着铲子，或正拿着一个还未种下的种子，则点击无效。
  2. 如果向日葵种子正处于冷却状态，则点击无效。（如果你实现了冷却遮盖物，则遮盖物会遮挡住点击）
  3. 询问 GameWorld 是否有 50 阳光。如果有，则花费 50 阳光，进入冷却状态，在原地（自身 x，y 坐标位置）生成一个冷却遮盖物，并在下一次点击种植位时种下向日葵。
     - 花费阳光后，你并不能直接生成向日葵，因为只有点击种植位后才能知道它应当生成在哪里。
     - 那么，点击种子后要做什么呢？（没有想到的话，再去读读上一页）

向日葵(Sunflower)

- **当被创建时**
  - 向日葵的贴图编号为 IMGID_SUNFLOWER。
  - 向日葵的位置并不固定，而由创建它的代码决定。（因此，它的构造函数中必须包含此部分。）
  - 向日葵的所在层级为 LAYER_PLANTS。
  - 向日葵的宽度为 60，高度为 80。
  - 向日葵具有 ANIMID_IDLE_ANIM 动画。
  - 向日葵具有 300 点 HP。
  - 向日葵第一次产生阳光的时刻是[30, 600]的随机 int，之后产生阳光的间隔为 600 刻。
    - 也就是说，运气好的话，刚种下 1 秒多就可以获得阳光。
- **当 Update()时**
  1. 向日葵需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
  2. 如果向日葵在此刻产生阳光，它将在原地（自身 x，y 坐标处）生成一个“由向日葵产生的阳光”。
- **当被点击时**

如果玩家正拿着铲子，那么向日葵需要死亡。放下铲子。

◆ 冷却遮盖物(Cooldown Mask)

◆ **当被创建时**

- 冷却遮盖物的贴图编号为 IMGID_COOLDOWN_MASK。
- 冷却遮盖物的位置由创建它的代码决定。
- 冷却遮盖物的所在层级为 LAYER_COOLDOWN_MASK。(会遮挡种子)
- 冷却遮盖物的宽度为 50，高度为 70。
- 冷却遮盖物不具有动画。
- 冷却遮盖物需要在对应的种子冷却后消失。

◆ **当 Update()时**

冷却遮盖物需要在对应的种子冷却后消失。

◆ **当被点击时**

冷却遮盖物什么也不需要做。

◆ 豌豆射手种子(Peashooter Seed)

◆ **当被创建时**

- 豌豆射手种子的贴图编号为 IMGID_SEED_PEASHOOTER。
- 豌豆射手种子位于第 2 个种子包位置，每个种子包的横向间隔为 60。
- 豌豆射手种子的所在层级为 LAYER_UI。
- 豌豆射手种子的宽度为 50，高度为 70。
- 豌豆射手种子不具有动画。
- 豌豆射手种子的价格是 100 阳光。
- 豌豆射手种子的冷却时间是 240 个 tick（8 秒）。

◆ **当 Update()时**

豌豆射手种子的 Update()并没有必须要做的事情，然而，基于你的实现，你也可以在 Update()中做些什么……

◆ **当被点击时**

1. 如果玩家正拿着铲子，或正拿着一个还未种下的种子，则点击无效。

2. 如果豌豆射手种子正处于冷却状态，则点击无效。（如果你实现了冷却遮盖物，则遮盖物会遮挡住点击）

3. 询问 GameWorld 是否有 100 阳光。如果有，则花费 100 阳光，进入冷却状态，在原地（自身 x, y 坐标位置）生成一个冷却遮盖物，并在下一次点击种植位时种下豌豆射手。

◆ 豌豆射手(Peashooter)

◆ 当被创建时

- 豌豆射手的贴图编号为 IMGID_PEASHOOTER。
- 豌豆射手的位置由创建它的代码决定。
- 豌豆射手的所在层级为 LAYER_PLANTS。
- 豌豆射手的宽度为 60，高度为 80。
- 豌豆射手具有 ANIMID_IDLE_ANIM 动画。
- 豌豆射手具有 300 点 HP。
- 豌豆射手每 30 tick（1 秒）发射一颗豌豆。

◆ 当 Update()时

1. 豌豆射手需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
2. 如果豌豆射手的攻击能力正在冷却，则降低 1 冷却时间，它的 Update()返回。
3. 如果豌豆射手可以攻击，它将询问 GameWorld，自己的一行上，自己的右边是否存在僵尸。若存在，则在豌豆射手自身坐标的右方 30 像素、上方 20 像素位置生成一颗豌豆，然后进入 30 tick 的冷却时间。

◆ 当被点击时

如果玩家正拿着铲子，那么豌豆射手需要死亡。放下铲子。

◆ 豌豆(Pea)

◆ 当被创建时

- 豌豆的贴图编号为 IMGID_PEA。
- 豌豆的位置由创建它的代码决定。
- 豌豆的所在层级为 LAYER_PROJECTILES。
- 豌豆的宽度为 28，高度为 28。

- 豌豆不具有动画。

◆ **当 Update()时**

1. 豌豆需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
2. 豌豆将向右移动 8 像素。
3. 如果豌豆飞出了屏幕右边界(x >= WINDOW_WIDTH)，豌豆需要死亡。

◆ **当被点击时**

豌豆什么也不需要做。

◆ **当与僵尸碰撞时**

豌豆将会对与它碰撞的僵尸造成 20 点伤害，然后死亡。

◆ 坚果墙种子(Wallnut Seed)

◆ **当被创建时**

- 坚果墙种子的贴图编号为 IMGID_SEED_WALLNUT。
- 坚果墙种子位于第 3 个种子包位置，每个种子包的横向间隔为 60。
- 坚果墙种子的所在层级为 LAYER_UI。
- 坚果墙种子的宽度为 50，高度为 70。
- 坚果墙种子不具有动画。
- 坚果墙种子的价格是 50 阳光。
- 坚果墙种子的冷却时间是 900 个 tick（30 秒）。

◆ **当 Update()时**

坚果墙种子的Update()并没有必须要做的事情，然而，基于你的实现，你也可以在Update()中做些什么……

◆ **当被点击时**

1. 如果玩家正拿着铲子，或正拿着一个还未种下的种子，则点击无效。
2. 如果坚果墙种子正处于冷却状态，则点击无效。（如果你实现了冷却遮盖物，则遮盖物会遮挡住点击）
3. 询问 GameWorld 是否有 50 阳光。如果有，则花费 50 阳光，进入冷却状态，在原地（自身 x，y 坐标位置）生成一个冷却遮盖物，并在下一次点击种植位时种下坚果墙。

◆ 坚果墙(Wallnut)

◆ 当被创建时

- 坚果墙的贴图编号为 IMGID_WALLNUT。
- 坚果墙的位置由创建它的代码决定。
- 坚果墙的所在层级为 LAYER_PLANTS。
- 坚果墙的宽度为 60，高度为 80。
- 坚果墙具有 ANIMID_IDLE_ANIM 动画。
- 坚果墙具有 4000 点 HP。

◆ 当 Update()时

1. 坚果墙需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
2. 如果坚果墙的 HP 小于总 HP 的 1/3，将它的贴图更换为“受损的坚果墙” IMGID_WALLNUT_CRACKED。

◆ 当被点击时

如果玩家正拿着铲子，那么坚果墙需要死亡。放下铲子。

◆ 樱桃炸弹种子(Cherry Bomb Seed)

◆ 当被创建时

- 樱桃炸弹种子的贴图编号为 IMGID_SEED_CHERRY_BOMB。
- 樱桃炸弹种子位于第 4 个种子包位置，每个种子包的横向间隔为 60。
- 樱桃炸弹种子的所在层级为 LAYER_UI。
- 樱桃炸弹种子的宽度为 50，高度为 70。
- 樱桃炸弹种子不具有动画。
- 樱桃炸弹种子的价格是 150 阳光。
- 樱桃炸弹种子的冷却时间是 1200 个 tick（40 秒）。

◆ 当 Update()时

樱桃炸弹种子的 Update()并没有必须要做的事情，然而，基于你的实现，你也可以在 Update()中做些什么……

◆ 当被点击时

1. 如果玩家正拿着铲子，或正拿着一个还未种下的种子，则点击无效。
2. 如果樱桃炸弹种子正处于冷却状态，则点击无效。（如果你实现了冷却遮盖物，则遮盖物会遮挡住点击）
3. 询问 GameWorld 是否有 150 阳光。如果有，则花费 150 阳光，进入冷却状态，在原地（自身 x, y 坐标位置）生成一个冷却遮盖物，并在下一次点击种植位时种下樱桃炸弹。

◆ 樱桃炸弹(Cherry Bomb)

◆ 当被创建时

- 樱桃炸弹的贴图编号为 IMGID_CHERRY_BOMB。
- 樱桃炸弹的位置由创建它的代码决定。
- 樱桃炸弹的所在层级为 LAYER_PLANTS。
- 樱桃炸弹的宽度为 60，高度为 80。
- 樱桃炸弹具有 ANIMID_IDLE_ANIM 动画。
- 樱桃炸弹具有 4000 点 HP。（以保证它不会在爆炸前被僵尸吃掉）

◆ 当 Update()时

樱桃炸弹需要在种下后的第 15 帧死亡，并在原地生成一个爆炸特效。

◆ 当被点击时

如果玩家正拿着铲子，那么樱桃炸弹需要死亡。放下铲子。

◆ 爆炸(Explosion)

◆ 当被创建时

- 爆炸的贴图编号为 IMGID_EXPLOSION。
- 爆炸的位置由创建它的代码决定。
- 爆炸的所在层级为 LAYER_PROJECTILES。
- 爆炸的宽度为 3 * LAWN_GRID_WIDTH，高度为 3 * LAWN_GRID_HEIGHT。
- 爆炸不具有动画。
- 爆炸的存在时间为 3 帧。

◆ 当 Update()时

爆炸需要在被创建的 3 帧后消失。

◆ **当被点击时**

爆炸什么都不需要做。

◆ **当与僵尸碰撞时**

在爆炸存在的 3 帧内，它将杀死所有与它碰撞的僵尸。

◆ 双发射手种子(Repeater Seed)

◆ **当被创建时**

- 双发射手种子的贴图编号为 IMGID_SEED_REPEATER。
- 双发射手种子位于第 5 个种子包位置，每个种子包的横向间隔为 60。
- 双发射手种子的所在层级为 LAYER_UI。
- 双发射手种子的宽度为 50，高度为 70。
- 双发射手种子不具有动画。
- 双发射手种子的价格是 200 阳光。
- 双发射手种子的冷却时间是 240 个 tick（8 秒）。

◆ **当 Update()时**

双发射手种子的 Update()并没有必须要做的事情，然而，基于你的实现，你也可以在 Update()中做些什么……

◆ **当被点击时**

1. 如果玩家正拿着铲子，或正拿着一个还未种下的种子，则点击无效。
2. 如果双发射手种子正处于冷却状态，则点击无效。(如果你实现了冷却遮盖物，则遮盖物会遮挡住点击)
3. 询问 GameWorld 是否有 200 阳光。如果有，则花费 200 阳光，进入冷却状态，在原地（自身 x，y 坐标位置）生成一个冷却遮盖物，并在下一次点击种植位时种下双发射手。

◆ 双发射手(Repeater)

◆ **当被创建时**

- 双发射手的贴图编号为 IMGID_REPEATER。
- 双发射手的位置由创建它的代码决定。
- 双发射手的所在层级为 LAYER_PLANTS。

- 双发射手的宽度为 60，高度为 80。
- 双发射手具有 ANIMID_IDLE_ANIM 动画。
- 双发射手具有 300 点 HP。
- 双发射手每 30 tick（1 秒）发射两颗豌豆。

◆ **当 Update()时**

1. 双发射手需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
2. 如果双发射手的攻击能力正在冷却，则降低 1 冷却时间，它的 Update()返回。
3. 如果双发射手可以攻击，它将询问 GameWorld，自己的一行上，自己的右边是否存在僵尸。若存在，则在双发射手的右方 30 像素、上方 20 像素位置生成一颗豌豆，进入 30 tick 的冷却时间，在之后的第 5 tick 再次在同一位置生成一颗豌豆。
    - ◆ 换言之，在发射第二发豌豆后，双发射手的攻击能力还需冷却 25 tick。

◆ **当被点击时**

如果玩家正拿着铲子，那么双发射手需要死亡。放下铲子。

◆ 铲子(Shovel)

◆ **当被创建时**

- 铲子的贴图编号为 IMGID_SHOVEL。
- 铲子位于(x = 600, y = WINDOW_HEIGHT − 40)。
- 铲子的所在层级为 LAYER_UI。
- 铲子的宽度为 50，高度为 50。
- 铲子不具有动画。

◆ **当 Update()时**

铲子什么也不需要做。

◆ **当被点击时**

1. 如果玩家正拿着一个还未种下的种子，则点击无效。
2. 如果玩家正拿着铲子，则放下铲子。

◆ 普通僵尸(Regular Zombie)

◆ 当被创建时

- 普通僵尸的贴图编号为 IMGID_REGULAR_ZOMBIE。
- 普通僵尸的位置由创建它的代码决定。
- 普通僵尸的所在层级为 LAYER_ZOMBIES。
- 普通僵尸的宽度为 20，高度为 80，如图。这样设定是为了防止它同时与两棵植物碰撞。
- 普通僵尸初始具有 ANIMID_WALK_ANIM 动画。
- 普通僵尸具有 200 点 HP。

◆ 当 Update()时：

1. 普通僵尸需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
2. 若普通僵尸当前正在行走，它将向左移动 1 像素。若普通僵尸当前正在啃食植物，它将不会移动。

◆ 当被点击时

普通僵尸什么也不需要做。

◆ 当发生碰撞时

- 如果与一颗豌豆发生碰撞，普通僵尸将会受到 20 点伤害，并让那颗豌豆死亡。
- 如果与爆炸发生碰撞，普通僵尸将会立即死亡。
- 如果与任何植物发生碰撞，普通僵尸将会：
  - 如果当前正在行走，则进入啃食状态，并播放 ANIMID_EAT_ANIM 动画。
  - 对那棵植物造成 3 点伤害。(这是每帧 3 点的持续伤害，即每秒 90 点)

◆ 铁桶僵尸(Bucket Head Zombie)

◆ 当被创建时

- 铁桶僵尸的贴图编号为 IMGID_BUCKET_HEAD_ZOMBIE。
- 铁桶僵尸的位置由创建它的代码决定。
- 铁桶僵尸的所在层级为 LAYER_ZOMBIES。
- 铁桶僵尸的宽度为 20，高度为 80。

- 铁桶僵尸初始具有 ANIMID_WALK_ANIM 动画。
- 铁桶僵尸具有 1300 点 HP。（铁桶 1100 点，僵尸 200 点）

◆ **当 Update()时：**

1. 铁桶僵尸需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。
2. 若铁桶僵尸当前正在行走，它将向左移动 1 像素。若铁桶僵尸当前正在啃食植物，它将不会移动。
3. 若铁桶僵尸的 HP 小于等于 200 点，它将失去铁桶。将它的贴图更换为普通僵尸的贴图。

◆ **当被点击时**

铁桶僵尸什么也不需要做。

◆ **当发生碰撞时**

- 如果与一颗豌豆发生碰撞，铁桶僵尸将会受到 20 点伤害，并让那颗豌豆死亡。
- 如果与爆炸发生碰撞，铁桶僵尸将会立即死亡。
- 如果与任何植物发生碰撞，铁桶僵尸将会：
  - 如果当前正在行走，则进入啃食状态，并播放 ANIMID_EAT_ANIM 动画。
  - 对那棵植物造成 3 点伤害。（这是每帧 3 点的持续伤害，即每秒 90 点）

◆ 撑杆跳僵尸(Pole Vaulting Zombie)

撑杆跳僵尸是《植物大战僵尸》的设计师 George Fan 最喜欢的一种僵尸，因为玩家第一次遇见它时常常会发生很有趣的节目效果：试着用坚果墙挡住它却被跳了过去。

◆ **当被创建时**

- 撑杆跳僵尸的贴图编号为 IMGID_POLE_VAULTING_ZOMBIE。
- 撑杆跳僵尸的位置由创建它的代码决定。
- 撑杆跳僵尸的所在层级为 LAYER_ZOMBIES。
- 撑杆跳僵尸的宽度为 20，高度为 80。
- 撑杆跳僵尸初始具有 ANIMID_RUN_ANIM 动画。
- 撑杆跳僵尸具有 340 点 HP。

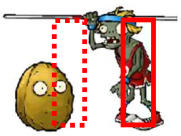
◆ **当 Update()时：**

1. 撑杆跳僵尸需要首先检查自己是否已经死亡。若已经死亡，它将等待 GameWorld 清理。它的 Update()应当立刻返回，无视下述步骤。

2. 若撑杆跳僵尸当前正在奔跑，它将检测左方 40 像素位置是否有可以跳过的植物：

   a) 它会在这一 tick，暂时将自己向左移动 40 像素。

   b) 询问 GameWorld，它是否在与一棵植物碰撞。如果是，它将：



      - 停止移动（为了和动画保持一致），播放 ANIMID_JUMP_ANIM 动画。在 42 帧后转为行走(ANIMID_WALK_ANIM)并向左“瞬移”150 像素（还是为了和动画保持一致，如右图所示）

   c) 无论它是否在这一帧触发了 b)，不要忘了再向右移回 40 像素以撤回你在 a)中的临时修改。

3. 若撑杆跳僵尸当前正在奔跑，它将向左移动 2 像素。若撑杆跳僵尸当前正在行走，它将向左移动 1 像素。若撑杆跳僵尸当前正在啃食植物或跳过植物，它将不会移动。

◆ **当被点击时**

撑杆跳僵尸什么也不需要做。

◆ **当发生碰撞时**

- 如果与一颗豌豆发生碰撞，撑杆跳僵尸将会受到 20 点伤害，并让那颗豌豆死亡。
- 如果与爆炸发生碰撞，撑杆跳僵尸将会立即死亡。
- 如果与任何植物发生碰撞，撑杆跳僵尸将会：
  - 如果当前正在行走，则进入啃食状态，并播放 ANIMID_EAT_ANIM 动画。
  - 对那棵植物造成 3 点伤害。（这是每帧 3 点的持续伤害，即每秒 90 点）

◆ 附录 – 碰撞处理

每个对象的碰撞体均为中心在`(x, y)`，横向宽度为 `width`，纵向高度为 `height` 的长方形。

若两对象的碰撞体长方形发生重叠，则认为两对象发生碰撞。

提示：碰撞关系似乎看起来有些复杂：有可能左右或上下交叉，有可能一个包围另一个，但实际上有非常简单的判断方法：考虑两个对象的中心点，是否碰撞是否跟他们中心点之间的 `x` 或 `y` 距离有关？