表格

描述已自动生成

这部分作业要求你实现一个“游戏暂停”功能，具体要求如下：

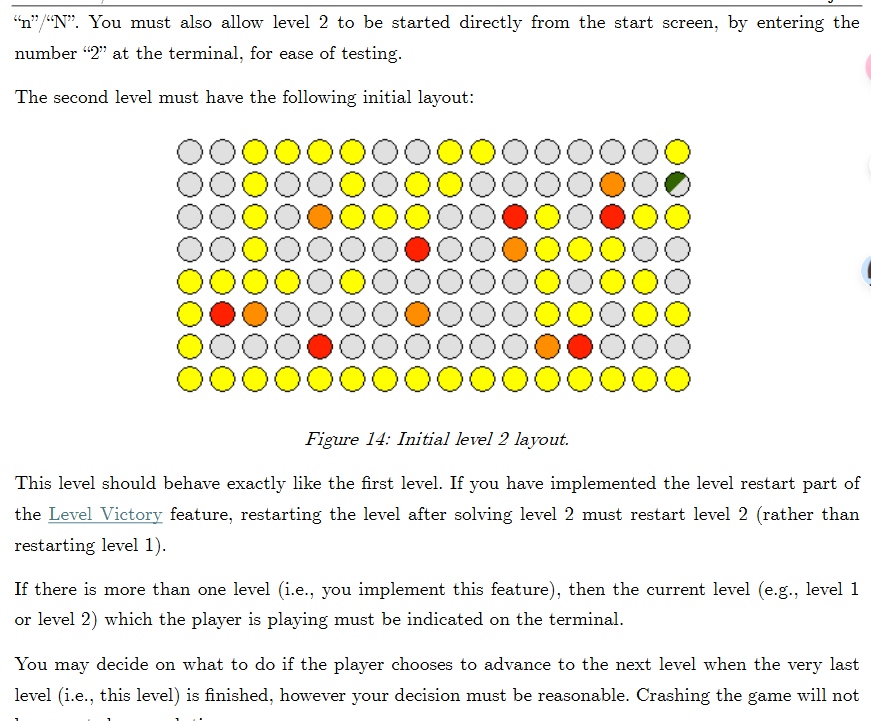
1. **按键暂停功能**：
   * 当在游戏中按下“p”或“P”键时（不包括开始画面或游戏结束画面），游戏应该暂停。
   * 当按下“p”或“P”键再次被按下时，游戏继续。
   * 在暂停时，终端上应该显示游戏已经暂停的提示信息（不要显示在消息区域，而是显示在屏幕上）。
2. **忽略其他输入**：
   * 在游戏暂停期间，除“p”/“P”键外的所有输入都应被忽略（包括按钮按下）。
3. **玩家图标的闪烁状态**：
   * 在暂停期间，玩家图标应该停止闪烁。
   * 在恢复游戏时，玩家图标的闪烁状态应该继续。例如，如果玩家图标在暂停时已经闪烁了120毫秒，那么恢复后它应该在80毫秒后熄灭。不能在恢复后立即闪烁或重置闪烁状态，确保它在恢复后继续完成之前的闪烁周期。

**实现方法：**

1. **按键检测**：
   * 需要在游戏主循环中增加对“p”/“P”键的检测，一旦检测到该按键，就进入暂停状态。
2. **暂停逻辑**：
   * 进入暂停状态后，显示一个“游戏暂停”的消息，直到再次按下“p”/“P”键。
   * 在暂停状态下，忽略其他所有输入，这意味着在循环中需要检查输入是否为“p”/“P”，如果不是则跳过处理。
3. **闪烁状态处理**：
   * 当暂停时，停止玩家图标的闪烁计时器。
   * 恢复时，需要保留闪烁的剩余时间，而不是重置计时。

在 Microchip Studio 中，你可以通过修改现有的游戏循环逻辑，在检测到按下暂停键时进入暂停状态，然后等待再次按下暂停键来恢复游戏。在暂停期间，你需要确保不处理其他按钮的输入，同时停止图标的闪烁。

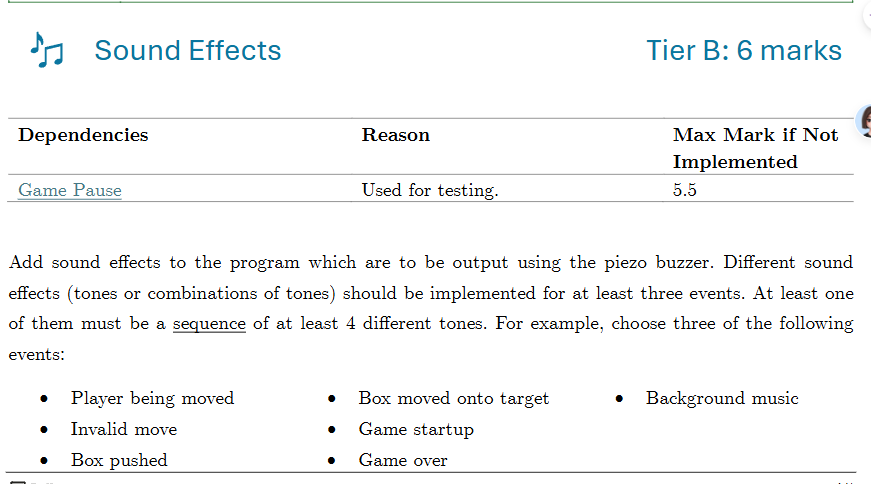
文本

描述已自动生成

这部分任务是要实现一个“第二关”的功能，并允许玩家在完成第一关后通过按下“n”或“N”键进入第二关。以下是实现步骤的详细说明：

**实现步骤：**

1. **扩展“游戏胜利”功能**：
   * 在你之前实现的“游戏胜利（Level Victory）”功能中，检测到游戏结束时，加入一个选项：如果玩家按下“n”或“N”键，进入第二关。
   * 例如，当第一关完成后，等待玩家按下按键来选择是重玩第一关还是进入第二关。
2. **在游戏结束处理函数中加入逻辑**：
   * 在 handle\_game\_over() 函数中（通常位于 project.c 文件中），增加逻辑来检测玩家输入“n”或“N”。如果玩家选择进入下一关，就加载第二关的初始布局。
3. **设置第二关的初始布局**：
   * 第二关的布局和第一关不同，需要你按照提供的图像中的布局来初始化。这意味着你需要在 initialise\_game() 函数中增加对不同关卡布局的处理逻辑。
   * 可以定义一个全局变量 current\_level，用于记录当前处于第几关，根据 current\_level 的值加载相应的布局。
4. **支持直接从起始画面启动第二关**：
   * 在 start\_screen() 函数中添加对输入“2”的处理。输入“2”时，可以直接开始第二关，而不是第一关。这主要用于测试的方便。
5. **重置或重新开始的逻辑**：
   * 如果玩家完成了第二关并想重新开始，你需要保证重新开始时能够正确地重置到对应关卡。例如，完成第二关后，重新开始应该直接重新进入第二关，而不是回到第一关。
   * 可以在 handle\_game\_over() 函数中根据玩家选择重新开始当前关卡，或者返回起始画面。
6. **显示当前关卡**：
   * 游戏过程中，可以在终端上显示当前关卡的编号，以帮助玩家了解当前处于哪个关卡。
   * 你可以在游戏开始时或者界面更新时，通过 printf() 函数在终端上显示类似“当前关卡：1”或“当前关卡：2”的提示信息。



文本

中度可信度描述已自动生成

这个部分要求你在程序中增加音效效果，使用压电蜂鸣器来输出不同的音效。这些音效应该针对至少三个不同的事件，其中至少一个事件需要是由至少4个不同音调组成的音效序列。例如，你可以选择以下三个事件来添加音效：

1. 玩家移动时
2. 无效移动时
3. 箱子被推动时
4. 箱子被移动到目标上时
5. 游戏启动时
6. 游戏结束时
7. 背景音乐

具体要求包括：

* 按下终端中的“q”或“Q”键时，应该切换游戏的静音状态，默认情况下，AVR微控制器上电时游戏应为非静音状态。
* 如果有实现背景音乐，它可以在游戏开始时或任何时候播放（比如游戏开始画面和游戏结束画面）。
* 你的特性总结表中必须指出哪些事件有不同的音效，并明确说明压电蜂鸣器连接到哪个引脚。音效必须为音调（而不是点击声），频率范围在20Hz到5kHz之间。
* 音效的存在与否不应影响游戏的运行速度。例如，游戏的播放速度在打开和关闭音效的情况下应该保持一致。
* 如果实现了“Game Pause”功能，游戏暂停期间不应播放任何音效，恢复游戏后音效应从暂停的地方继续。如果实现超过三个事件的音效，则不会获得额外的分数。

为实现这些功能，你可以：

1. 在AVR程序中，使用定时器生成PWM信号，以控制压电蜂鸣器发出不同频率的声音。
2. 使用一个状态变量来记录当前音效是否处于静音状态，然后在对应的事件发生时判断是否播放音效。
3. 根据游戏的事件，调用相应的音效函数来输出音调。
4. 当“q”或“Q”按下时，改变静音状态变量的值。

这些音效的实现能够有效地增加游戏的交互性和趣味性，同时也符合设计要求中的音频反馈部分。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

这部分要求你在游戏中为目标方块和箱子的交互添加视觉效果，包括闪烁和动画效果。以下是具体的实现方法：

1. **目标方块的闪烁效果**：
   * 目标方块需要以每秒一次的频率在红色和黑色之间闪烁，每种颜色显示500ms。闪烁效果需要和玩家图标的闪烁类似，只是速度更慢。
   * 当一个箱子放在目标上时，目标方块应保持绿色不变。
   * 当玩家图标位于目标上时，应在深绿色和红色之间以200ms的频率闪烁（每种颜色200ms）。

**实现步骤**：

* + 使用定时器来控制闪烁效果。比如，你可以设置一个定时器，每500ms改变目标方块的颜色（红色到黑色）。
  + 在箱子放到目标上时，取消闪烁效果并设置为绿色。
  + 玩家图标在目标上时，使用不同的颜色（深绿色和红色）进行较快的闪烁。

1. **箱子放到目标上的动画**：
   * 当箱子被推到目标位置时，显示一个动画，突出显示该目标位置。
   * 该动画必须至少持续500ms，同时在播放动画的过程中，所有游戏输入必须仍然正常工作。
   * 如果第二个箱子在第一个动画播放期间被推到另一个目标，你可以选择提前结束当前动画并开始新的动画，或者让它们重叠播放。
   * 如果箱子在动画播放期间被移动离开目标位置，动画需要提前结束。

**实现步骤**：

* + 使用一个定时器来控制动画的时长（至少500ms）。
  + 你可以通过改变目标方块的颜色、闪烁频率，或者在目标周围显示一些额外的图案来实现动画效果。
  + 处理多个动画重叠的情况，可以考虑使用一个队列来管理动画的播放顺序，或者让新动画替换当前动画。

1. **游戏暂停和动画的处理**：
   * 如果实现了“Game Pause”功能，当游戏暂停时，闪烁和动画效果也必须暂停，并在游戏恢复后继续。
   * 你可以使用一个状态变量来记录游戏是否处于暂停状态，如果是暂停状态，则不进行任何动画和闪烁的更新。

**具体实现代码的建议**：

* 使用AVR的定时器（比如Timer1或Timer2）来控制目标方块的闪烁和动画。定时器可以用来生成中断，每次中断时更新方块的显示状态。
* 在游戏逻辑中，添加对箱子位置的检查，当箱子放置到目标位置时，触发动画效果。
* 在暂停功能中，通过一个全局变量控制闪烁和动画的执行逻辑，确保在游戏暂停时所有的视觉效果都停止。

这些视觉效果能够提高游戏的视觉反馈，使游戏更加有趣和易于理解。可以考虑将这些逻辑模块化，便于代码的管理和调试。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

文本

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 电子邮件

描述已自动生成

关于如何实现“使用摇杆移动玩家”的功能，以下是一些建议：

**1. 摇杆输入的读取**

* 使用ADC（模数转换器）读取2轴摇杆的X轴和Y轴的电压值。通常，摇杆的中心位置会返回中等电压值（比如5V供电的情况下大约2.5V），而向上、向下、向左和向右移动时，电压会接近0V或5V。
* 根据读取的电压值，确定玩家是否想向上、下、左、右或对角移动。

**2. 玩家移动逻辑**

* 当摇杆方向确定后，调用现有的玩家移动函数（比如move\_player()），来执行移动操作。要确保玩家可以向8个方向移动（上、下、左、右、东北、西北、东南、西南）。
* 对于对角移动，需要分成两个步骤来完成，每次移动一个单元格。比如，如果玩家从当前位置向东北方向移动，应该先向上移动，再向右移动，或者反之。

**3. 对角移动的处理**

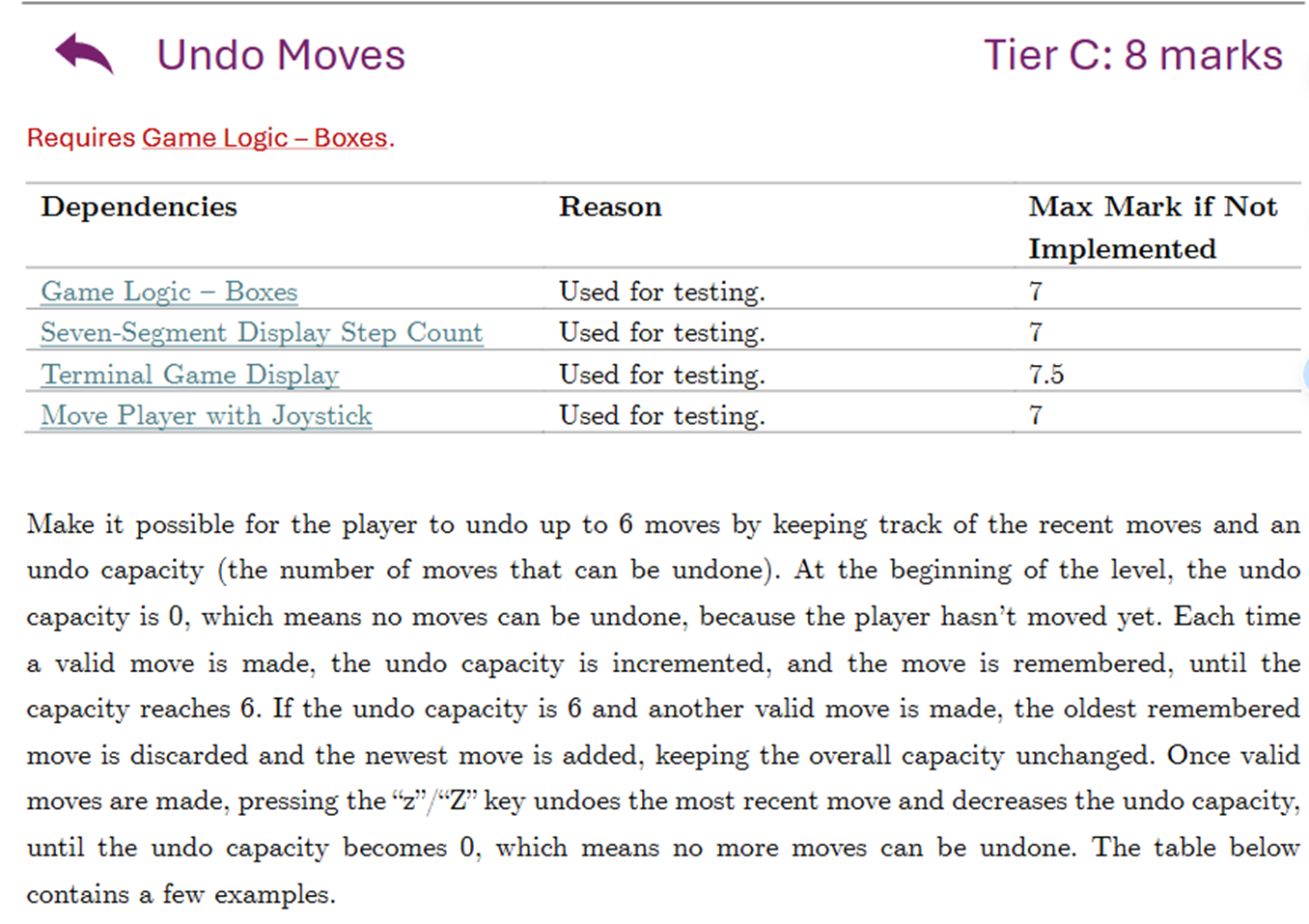
* 当进行对角移动时，需要分为两个方向的单独移动。确保步数增加为2（每个方向各1步）。
* 如果目标位置包含墙或箱子，则处理为无效移动并在终端显示相关信息。
  + 如果是墙，输出消息：无法向对角方向移动，因为前方有障碍物。
  + 如果是箱子，输出消息：无法对角推动箱子。

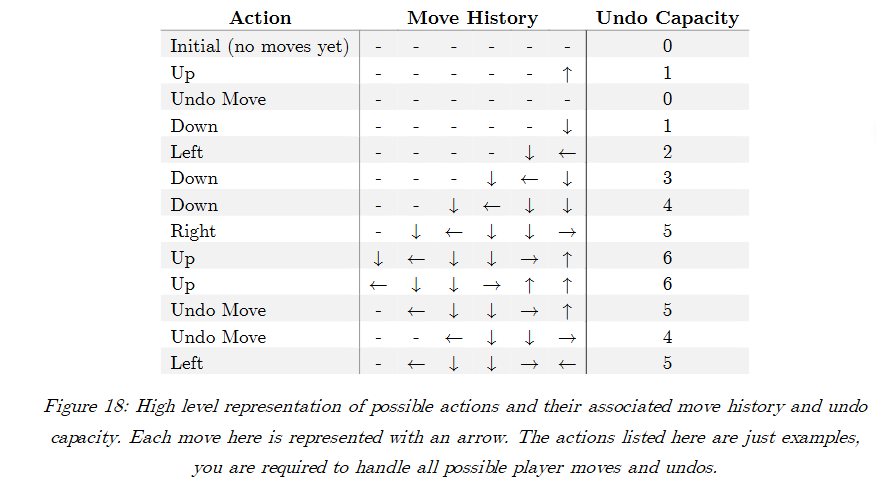
**4. 障碍物的处理**

* 在执行对角移动之前，检查目标路径上是否有墙或箱子。如果其中一个方向上有障碍，则只能执行另一个方向的单步移动，并在终端显示适当的消息。
* 如果两个方向上都有障碍，则移动无效，输出对应的错误信息。

**5. 重复移动**

* 当用户长按摇杆时，玩家应持续移动。你可以自行决定重复的速度，但要确保速度不会太快以致游戏不可玩，或者太慢以致用户感到不耐烦。
* 当执行自动重复移动时，其他输入（如按钮或终端输入）也应继续有效。





文本

低可信度描述已自动生成

表格

描述已自动生成

这个功能是为游戏添加“撤销移动”的功能。你需要实现一个最多可以撤销6次移动的机制，这意味着玩家可以在一定范围内返回到之前的位置。

具体的实现步骤：

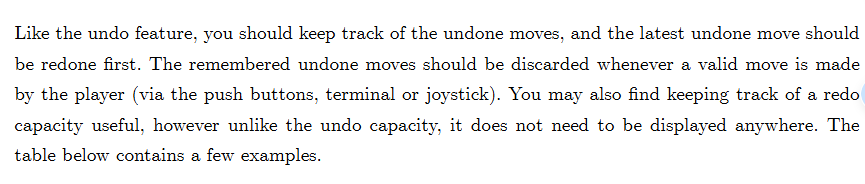
1. **移动记录与撤销容量**：每次有效移动时，你需要将移动存入一个记录中，直到记录的容量达到6次。如果有新的移动被记录（当容量已满时），你需要删除最早的移动并将新移动添加到记录中。撤销容量反映了当前可以撤销的次数，从0开始，逐渐增加到最大值6。
2. **撤销逻辑**：
   * 当玩家按下“Z/z”键时，执行最近一次移动的撤销，并将撤销容量减少1。
   * 在每次撤销时，需要恢复移动前的状态，包括玩家的位置、箱子的状态等。这意味着你需要记录每次移动的细节，包含玩家和箱子的具体位置。
3. **LED指示撤销容量**：使用LED L0到L5来表示当前的撤销容量。例如，如果撤销容量是0，则没有LED亮起；如果容量为6，则所有LED亮起。当容量增加或减少时，相应地点亮或熄灭LED。
4. **细节**：
   * 撤销移动的效果要反映在LED矩阵、七段显示器和终端显示上，恢复到移动前的状态。
   * 如果玩家的移动涉及到推动箱子，撤销时也需要将箱子移回原位置。
   * 如果实现了“七段显示器步数计数”功能，撤销时步数也要相应减少。

**实现建议：**

* 使用一个数组（或环形缓冲区）来存储最近的6次移动，包括玩家和箱子的位置。
* 在每次移动后，将移动记录添加到数组中，并更新撤销容量。
* 在用户按下“Z/z”键时，从数组中读取最近的移动，恢复之前的状态，同时减少撤销容量。
* 使用LED显示撤销容量的大小，方便玩家知道当前还可以撤销多少次移动。

图形用户界面, 文本, 应用程序

描述已自动生成



图形用户界面, 应用程序, 表格

描述已自动生成

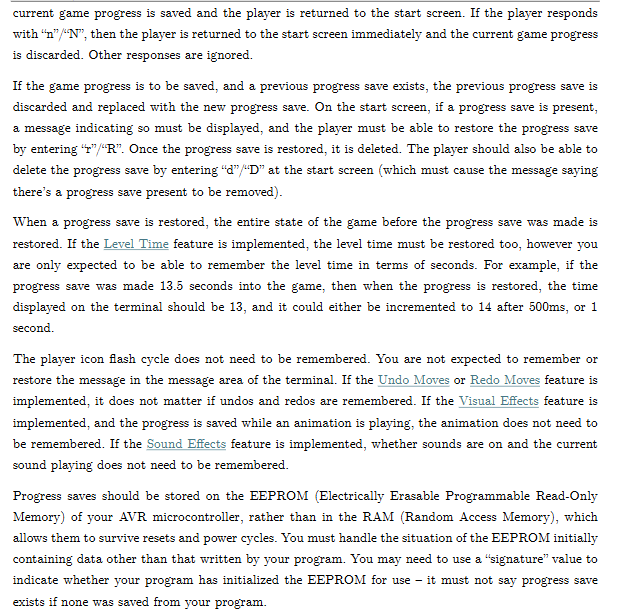
这个功能要求你实现“重做”最近撤销的动作（Redo），类似于文本编辑器中的“撤销/重做”功能。具体步骤如下：

1. **实现重做的逻辑**：
   * 当玩家按下“Y”键时，系统应该把最后一个被撤销的动作重做出来。
   * 重做的动作同样应该能够被撤销，因此需要把重做的动作重新存入“可撤销的动作历史”中。
   * 重做的动作只能在玩家没有进行任何新动作的情况下进行。一旦玩家在撤销动作之后进行了一个新动作，重做的历史将被清空。
2. **保存撤销和重做的历史**：
   * 你需要像实现撤销功能一样追踪撤销的动作，并在玩家按下“Y”键时执行重做动作。
   * 当玩家进行新动作时，需要清空之前保存的所有重做动作。
3. **界面反馈**：
   * 如果玩家没有进行任何撤销，或者玩家已经进行了新动作，那么按下“Y”键时什么都不应该发生，也可以选择在消息区域显示一条信息。
   * LED 依然用于显示撤销的容量，但不需要专门为重做动作显示容量。
4. **实现要点**：
   * 当重做动作发生时，游戏的各个元素（如LED矩阵显示、7段数码管、终端）应该恢复到重做之前的状态。
   * 重做时，保留原来撤销的历史，确保重做动作同样可以被再次撤销。
5. **注意事项**：
   * 像撤销功能一样，重做的动作应该是从最近的撤销动作开始执行。
   * 如果玩家在进行新的有效动作后按下了“Y”键，应该无效，可以选择在消息区提示信息。

**实现提示**： 你可以使用两个栈（或队列）来分别保存“撤销的动作”和“重做的动作”。当玩家进行撤销时，将动作从“撤销栈”移动到“重做栈”；当玩家进行重做时，将动作从“重做栈”移动回“撤销栈”。如果玩家进行了新的动作，重做栈则清空，以确保只能对最近撤销的动作进行重做。

图形用户界面, 文本, 应用程序

描述已自动生成



文本, Word

描述已自动生成

这个“Game Progress Save”功能需要在游戏暂停时扩展“Game Pause”功能，让玩家可以选择保存当前的游戏进度或返回开始屏幕，并允许从保存的位置恢复游戏。

**实现步骤：**

1. **游戏暂停并退出到开始屏幕**：
   * 在游戏暂停时，玩家按下“x”/“X”键，程序应在终端上提示玩家是否保存游戏进度。
   * 如果玩家选择“y”/“Y”，游戏进度会被保存，并返回开始屏幕。
   * 如果玩家选择“n”/“N”，则直接返回开始屏幕，不保存当前游戏进度。
2. **保存和恢复游戏进度**：
   * 当保存游戏进度时，如果已有之前的进度保存，则替换成新的进度保存，并显示一条消息告诉玩家已保存。
   * 在开始屏幕上，如果存在已保存的进度，玩家可以按下“r”/“R”键恢复保存的进度。恢复后，保存被删除。
   * 玩家也可以在开始屏幕按下“d”/“D”键删除保存的进度。
3. **保存的内容**：
   * 当保存的进度被恢复时，游戏的状态需要完全恢复，包括玩家的位置、箱子的位置、步数等。
   * 如果实现了“Level Time”功能，恢复时需要恢复计时器显示的时间（只需保存秒级的时间）。例如，保存时游戏进行到13.5秒，恢复后应显示13秒，随后每秒递增。
4. **其他状态**：
   * 需要注意的是，不需要保存玩家图标的闪烁周期状态，恢复时可以从初始状态开始。
   * 如果实现了“Undo Moves”或“Redo Moves”功能，不要求保存撤销/重做的状态。
   * 如果实现了“Visual Effects”功能，恢复时不需要记住是否有动画正在播放。
   * 如果实现了“Sound Effects”功能，不要求记住声音状态（如背景音乐是否播放）。
5. **进度保存存储方式**：
   * 进度保存应该存储在EEPROM（电可擦写编程只读存储器）中，而不是RAM，这样可以在系统重置或断电后保留进度。
   * 需要处理EEPROM初始状态为空的情况，可以用一个“签名值”来判断EEPROM是否被初始化。

**提示：**

* EEPROM存储是为了保证游戏进度可以在设备断电后依然保存。
* 当进度恢复后，必须让游戏的所有状态完全恢复到保存时的状态（如LED矩阵、七段显示器、终端输出等）。
* 代码实现时要注意处理存储的数据以及玩家与系统的交互。

**代码实现建议：**

* 你可以通过编写EEPROM读写函数来存储和读取游戏的关键信息，例如玩家位置、关卡状态、步数、箱子位置等。
* 当玩家在开始屏幕时，可以用一个特定的按键（如“r”）来触发游戏进度的恢复。
* 在代码中可以定义数据结构来保存游戏的状态，并将其存储到EEPROM中。

这些步骤可以帮助你实现保存和恢复游戏进度的功能，从而增强游戏的可玩性和用户体验。