二、实验原理

2.1 游戏描述

推箱子游戏中会有一个封闭的围墙，围城一个不规则的多边形区域，人和箱子只能在这个区域内活动。区域内有一个人，几个箱子和目标点，使用方向键控制人的位置推动箱子到目标点即为成功。一次只能推动一个箱子，如果箱子到了死角则无法继续游戏。

2.2 抽象角色

从上述描述中我们可以抽象出游戏中包括以下角色：

1. 围墙：阻挡活动路线的区域。
2. 空间：可以行走和推动箱子活动的区域。
3. 人：操作对象。
4. 箱子
5. 目标点

其中人，箱子，目标点都应该初始化在空间区域，而围墙区域是不可以出现其他角色的。

2.3 抽象操作

玩推箱子这个游戏我们唯一能操作的就是人这个角色，我们使用方向键控制人上下左右移动，人的移动有两种情况，这两种情况我们都需要分别处理：

1. 自己移动
2. 推动箱子移动

此外，游戏中我们支持下面两个操作：

1. 撤销：撤销刚才的移动，使用回退键控制
2. 重做：重新执行刚被撤销的移动，使用空格键控制

综上，我们需要支持的键盘操作事件为上下左右四个键及回退键和空格键。在下一节pygame的实现中我们需要处理这六个按键操作事件。

三、开发准备

3.1 实验环境

为了能够在环境中使用pygame，在实验环境中打开 Xfce 终端，并输入以下命令来安装 pygame：

```
$ sudo pip3 install pygame
```

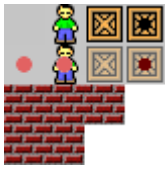
在Python中使用pygame，只需要加载：

```
import pygame
```

四、实验步骤

4.1 pygame 简单介绍

- pygame中的模块很多，包括对鼠标，显示设备，绘图，事件，字体，图片，键盘，声音，视频，音频等操作。在推箱子游戏中，我们将用到下面的模块：
    - pygame.display：访问显示设备，用来显示图像
    - pygame.image：加载和存储图片，用来处理雪碧图
    - pygame.key：读取键盘按键
    - pygame.event：管理事件，在游戏中处理键盘事件
    - pygame.time：管理时间和显示的帧信息
- 上面的介绍中提到了雪碧图，雪碧图是在游戏开发中常用的一种图像合并方法，将小图标和背景图像合并到一张图片上，然后利用pygame的图片定位来显示需要显示的图片部分。
- 在推箱子游戏中我们使用现成的雪碧图，这里不介绍如何切图及合成雪碧图，网上有无数的方法。
- 本项目中用到的推箱子雪碧图中的图像元素来自borgar网站。可以在下载的源代码中找到`borgar.png`这个文件，双击打开后可以看到里面包含游戏所需的所有效果图：



- 上图所示的游戏图像元素包括：
    - 游戏界面背景色
    - 人
    - 普通箱子
    - 目标点
    - 人和目标点重合效果
    - 箱子到达目标点重合效果
    - 围墙
- 其中有两个箱子图在我们的实现中不需要。pygame中使用`blit`方法加载并显示雪碧图中的内容，在后续实现部分我们会详细介绍。

4.2 实现框架

开始使用 pygame 把最初的游戏框架搭建起来

- 初始化pygame

```
pygame.init()
```

- 设置显示对象

```
# 设置pygame显示窗口大小为宽400，高300像素
screen = pygame.display.set_mode((400,300))
```

- 加载图像元素

```
#加载图像元素，所有图像元素都写在一个文件中
skinfilename = os.path.join('borgar.png')

try:
    skin = pygame.image.load(skinfilename)
except pygame.error as msg:
    print('cannot load skin')
    raise SystemExit(msg)

skin = skin.convert()

# 设置窗口显示的背景颜色，使用skin文件中坐标为(0,0)的元素
screen.fill(skin.get_at((0,0)))
```

- 设置时钟及键盘事件重复发生的时间 `key.set_repeat` 使用参数 `(delay, interval)` 调用设置重复事件发生的时间。

```
clock = pygame.time.Clock()
pygame.key.set_repeat(200,50)
```

- 启动主循环

```
# 游戏主循环
while True:
    clock.tick(60)
    pass
```

- 处理游戏事件及键盘操作。在主循环中我们需要处理键盘事件，在上面的内容中我们提到需要支持上下左右，回退及空格六个键。

```
# 获取游戏事件
for event in pygame.event.get():
    # 退出游戏操作事件
    if event.type == QUIT:
        pygame.quit()
        sys.exit()
    # 键盘操作
    elif event.type == KEYDOWN:
```

```
        # 向左移动
        if event.key == K_LEFT:
            pass
        # 向上移动
        elif event.key == K_UP:
            pass
        # 向右移动
        elif event.key == K_RIGHT:
            pass
        # 向下移动
        elif event.key == K_DOWN:
            pass
        # 撤销操作
        elif event.key == K_BACKSPACE:
            pass
        # 重做操作
        elif event.key == K_SPACE:
            pass
```

现在我们已经完成基于pygame的游戏框架，开始实现游戏的逻辑。

4.3 实现地图

首先我们需要定义sokoban对象，我们使用一个类来容纳所有游戏相关的逻辑。

```
class Sokoban:

    # 初始化推箱子游戏
    def __init__(self):
        pass
```

推箱子游戏需要一个操作的区域，即地图区域。我们使用一个字符列表来表示地图，列表中不同的字符表示游戏里不同的元素：

1. 围墙：`# 符号`
2. 空间：`- 符号`
3. 人：`@ 符号`
4. 箱子：`$ 符号`
5. 目标点：`. 符号`
6. 人和目标点重合：`+ 符号`
7. 箱子和目标点重合：`* 符号`

在游戏启动时我们需要先设置一个默认的地图字符列表，同时我们需要知道这个地图的宽度和高度，从而通过这个一维的列表生成2D的地图。

地图的表示类似于下面的代码，你是否能根据这些代码想象出启动后的样子：

```
class Sokoban:

    # 初始化推箱子游戏
    def __init__(self):
        # 设置地图
        self.level = list(
```

```
        self.level = list(
            "----#####----------"
            "----#---#----------"
            "----#$--#----------"
            "--###--$##---------"
            "--#--$-$-#---------"
            "###-#-##-#---######"
            "#---#-##-#####--..#"

            "#-$--$----------..#"
            "######-###-#@##--..#"
            "----#-----#########"
            "----#######--------")

        # 设置地图的宽度和高度以及人在地图中的位置（地图列表中的索引值）
        # 共19列
        self.w = 19

        # 共11行
        self.h = 11

        # 人的初始化位置在self.level[163]
        self.man = 163
```

地图的显示则会对字符列表进行扫描，根据不同的字符在对应的位置上显示不同的雪碧图中的元素。

由于显示为2D，所以扫描字符列表时会用到宽度和高度来判断每个字符在2D显示区域内的位置，我们需要将pygame中提到的screen和skin作为参数传给画图draw函数。

需要注意我们实现画图的函数为pygame的blit，这个函数将雪碧图中的图像提取出来显示在指定的位置：

```
screen.blit(skin, (i*w, j*w), (0,0,w,w))
```

完整的draw函数实现如下，首先进行扫描，然后根据字符显示雪碧图中对应的图像：

```

class Sokoban:

```

```
# 画图，根据地图level将内容显示到pygame的窗口中
def draw(self, screen, skin):

    # 获取每个图像元素的宽度
    w = skin.get_width() / 4

    # 遍历地图level中的每个字符元素
    for i in range(0, self.w):
        for j in range(0, self.h):

            # 获取地图中的第j行第i列
            item = self.level[j*self.w + i]

            # 该位置显示为墙#
            if item == "#":
```

```
            if item == '#':
                # 使用pygame的blit方法将图像显示到指定位置，
                # 位置坐标为(i*w, j*w)，图像在skin中的坐标及长宽为(0,2*w,w,w)
                screen.blit(skin, (i*w, j*w), (0,2*w,w,w))
            # 该位置显示为空间-
            elif item == '-':
                screen.blit(skin, (i*w, j*w), (0,0,w,w))
            # 该位置显示为人：@
            elif item == '@':
                screen.blit(skin, (i*w, j*w), (w,0,w,w))
            # 该位置显示为箱子：$
            elif item == '$':
                screen.blit(skin, (i*w, j*w), (2*w,0,w,w))
            # 该位置显示为目标点：.
            elif item == '.':
                screen.blit(skin, (i*w, j*w), (0,w,w,w))
            # 该位置显
```