俄罗斯方块游戏

#!/bin/bash

# Tetris Game

# 10.21.2003 xhchen< [email]xhchen@winbond.com.tw[/email]>

#APP declaration

APP\_NAME="${0##\*[\\/]}"

APP\_VERSION="1.0"

#颜色定义

cRed=1

cGreen=2

cYellow=3

cBlue=4

cFuchsia=5

cCyan=6

cWhite=7

colorTable=($cRed $cGreen $cYellow $cBlue $cFuchsia $cCyan $cWhite)

#位置和大小

iLeft=3

iTop=2

((iTrayLeft = iLeft + 2))

((iTrayTop = iTop + 1))

((iTrayWidth = 10))

((iTrayHeight = 15))

#颜色设置

cBorder=$cGreen

cScore=$cFuchsia

cScoreValue=$cCyan

#控制信号

#改游戏使用两个进程，一个用于接收输入，一个用于游戏流程和显示界面;

#当前者接收到上下左右等按键时，通过向后者发送signal的方式通知后者。

sigRotate=25

sigLeft=26

sigRight=27

sigDown=28

sigAllDown=29

sigExit=30

#七中不同的方块的定义

#通过旋转，每种方块的显示的样式可能有几种

box0=(0 0 0 1 1 0 1 1)

box1=(0 2 1 2 2 2 3 2 1 0 1 1 1 2 1 3)

box2=(0 0 0 1 1 1 1 2 0 1 1 0 1 1 2 0)

box3=(0 1 0 2 1 0 1 1 0 0 1 0 1 1 2 1)

box4=(0 1 0 2 1 1 2 1 1 0 1 1 1 2 2 2 0 1 1 1 2 0 2 1 0 0 1 0 1 1 1 2)

box5=(0 1 1 1 2 1 2 2 1 0 1 1 1 2 2 0 0 0 0 1 1 1 2 1 0 2 1 0 1 1 1 2)

box6=(0 1 1 1 1 2 2 1 1 0 1 1 1 2 2 1 0 1 1 0 1 1 2 1 0 1 1 0 1 1 1 2)

#所有其中方块的定义都放到box变量中

box=(${box0[@]} ${box1[@]} ${box2[@]} ${box3[@]} ${box4[@]} ${box5[@]} ${box6[@]})

#各种方块旋转后可能的样式数目

countBox=(1 2 2 2 4 4 4)

#各种方块再box数组中的偏移

offsetBox=(0 1 3 5 7 11 15)

#每提高一个速度级需要积累的分数

iScoreEachLevel=50 #be greater than 7

#运行时数据

sig=0 #接收到的signal

iScore=0 #总分

iLevel=0 #速度级

boxNew=() #新下落的方块的位置定义

cBoxNew=0 #新下落的方块的颜色

iBoxNewType=0 #新下落的方块的种类

iBoxNewRotate=0 #新下落的方块的旋转角度

boxCur=() #当前方块的位置定义

cBoxCur=0 #当前方块的颜色

iBoxCurType=0 #当前方块的种类

iBoxCurRotate=0 #当前方块的旋转角度

boxCurX=-1 #当前方块的x坐标位置

boxCurY=-1 #当前方块的y坐标位置

iMap=() #背景方块图表

#初始化所有背景方块为-1, 表示没有方块

for ((i = 0; i < iTrayHeight \* iTrayWidth; i++)); do iMap[$i]=-1; done

#接收输入的进程的主函数

function RunAsKeyReceiver()

{

local pidDisplayer key aKey sig cESC sTTY

pidDisplayer=$1

aKey=(0 0 0)

cESC=`echo -ne "\033"`

cSpace=`echo -ne "\040"`

#保存终端属性。在read -s读取终端键时，终端的属性会被暂时改变。

#如果在read -s时程序被不幸杀掉，可能会导致终端混乱，

#需要在程序退出时恢复终端属性。

sTTY=`stty -g`

#捕捉退出信号

trap "MyExit;" INT TERM

trap "MyExitNoSub;" $sigExit

#隐藏光标

echo -ne "\033[?25l"

while :

do

#读取输入。注-s不回显，-n读到一个字符立即返回

read -s -n 1 key

aKey[0]=${aKey[1]}

aKey[1]=${aKey[2]}

aKey[2]=$key

sig=0

#判断输入了何种键

if [[ $key == $cESC && ${aKey[1]} == $cESC ]]

then

#ESC键

MyExit

elif [[ ${aKey[0]} == $cESC && ${aKey[1]} == "[" ]]

then

if [[ $key == "A" ]]; then sig=$sigRotate #<向上键>

elif [[ $key == "B" ]]; then sig=$sigDown #< 向下键>

elif [[ $key == "D" ]]; then sig=$sigLeft #< 向左键>

elif [[ $key == "C" ]]; then sig=$sigRight #< 向右键>

fi

elif [[ $key == "W" || $key == "w" ]]; then sig=$sigRotate #W, w

elif [[ $key == "S" || $key == "s" ]]; then sig=$sigDown #S, s

elif [[ $key == "A" || $key == "a" ]]; then sig=$sigLeft #A, a

elif [[ $key == "D" || $key == "d" ]]; then sig=$sigRight #D, d

elif [[ "[$key]" == "[]" ]]; then sig=$sigAllDown #空格键

elif [[ $key == "Q" || $key == "q" ]] #Q, q

then

MyExit

fi

if [[ $sig != 0 ]]

then

#向另一进程发送消息

kill -$sig $pidDisplayer

fi

done

}

#退出前的恢复

function MyExitNoSub()

{

local y

#恢复终端属性

stty $sTTY

((y = iTop + iTrayHeight + 4))

#显示光标

echo -e "\033[?25h\033[${y};0H"

exit

}

function MyExit()

{

#通知显示进程需要退出

kill -$sigExit $pidDisplayer

MyExitNoSub

}

#处理显示和游戏流程的主函数

function RunAsDisplayer()

{

local sigThis

InitDraw

#挂载各种信号的处理函数

trap "sig=$sigRotate;" $sigRotate

trap "sig=$sigLeft;" $sigLeft

trap "sig=$sigRight;" $sigRight

trap "sig=$sigDown;" $sigDown

trap "sig=$sigAllDown;" $sigAllDown

trap "ShowExit;" $sigExit

while :

do

#根据当前的速度级iLevel不同，设定相应的循环的次数

for ((i = 0; i < 21 - iLevel; i++))

do

sleep 0.02

sigThis=$sig

sig=0

#根据sig变量判断是否接受到相应的信号

if ((sigThis == sigRotate)); then BoxRotate; #旋转

elif ((sigThis == sigLeft)); then BoxLeft; #左移一列

elif ((sigThis == sigRight)); then BoxRight; #右移一列

elif ((sigThis == sigDown)); then BoxDown; #下落一行

elif ((sigThis == sigAllDown)); then BoxAllDown; #下落到底

fi

done

#kill -$sigDown $$

BoxDown #下落一行

done

}

#BoxMove(y, x), 测试是否可以把移动中的方块移到(x, y)的位置, 返回0则可以, 1不可以

function BoxMove()

{

local j i x y xTest yTest

yTest=$1

xTest=$2

for ((j = 0; j < 8; j += 2))

do

((i = j + 1))

((y = ${boxCur[$j]} + yTest))

((x = ${boxCur[$i]} + xTest))

if (( y < 0 || y >= iTrayHeight || x < 0 || x >= iTrayWidth))

then

#撞到墙壁了

return 1

fi

if ((${iMap[y \* iTrayWidth + x]} != -1 ))

then

#撞到其他已经存在的方块了

return 1

fi

done

return 0;

}

#将当前移动中的方块放到背景方块中去,

#并计算新的分数和速度级。(即一次方块落到底部)

function Box2Map()

{

local j i x y xp yp line

#将当前移动中的方块放到背景方块中去

for ((j = 0; j < 8; j += 2))

do

((i = j + 1))

((y = ${boxCur[$j]} + boxCurY))

((x = ${boxCur[$i]} + boxCurX))

((i = y \* iTrayWidth + x))

iMap[$i]=$cBoxCur

done

#消去可被消去的行

line=0

for ((j = 0; j < iTrayWidth \* iTrayHeight; j += iTrayWidth))

do

for ((i = j + iTrayWidth - 1; i >= j; i--))

do

if ((${iMap[$i]} == -1)); then break; fi

done

if ((i >= j)); then continue; fi

((line++))

for ((i = j - 1; i >= 0; i--))

do

((x = i + iTrayWidth))

iMap[$x]=${iMap[$i]}

done

for ((i = 0; i < iTrayWidth; i++))

do

iMap[$i]=-1

done

done

if ((line == 0)); then return; fi

#根据消去的行数line计算分数和速度级

((x = iLeft + iTrayWidth \* 2 + 7))

((y = iTop + 11))

((iScore += line \* 2 - 1))

#显示新的分数

echo -ne "\033[1m\033[3${cScoreValue}m\033[${y};${x}H${iScore} "

if ((iScore % iScoreEachLevel < line \* 2 - 1))

then

if ((iLevel < 20))

then

((iLevel++))

((y = iTop + 14))

#显示新的速度级

echo -ne "\033[3${cScoreValue}m\033[${y};${x}H${iLevel} "

fi

fi

echo -ne "\033[0m"

#重新显示背景方块

for ((y = 0; y < iTrayHeight; y++))

do

((yp = y + iTrayTop + 1))

((xp = iTrayLeft + 1))

((i = y \* iTrayWidth))

echo -ne "\033[${yp};${xp}H"

for ((x = 0; x < iTrayWidth; x++))

do

((j = i + x))

if ((${iMap[$j]} == -1))

then

echo -ne " "

else

echo -ne "\033[1m\033[7m\033[3${iMap[$j]}m\033[4${iMap[$j]}m[]\033[0m"

fi

done

done

}

#下落一行

function BoxDown()

{

local y s

((y = boxCurY + 1)) #新的y坐标

if BoxMove $y $boxCurX #测试是否可以下落一行

then

s="`DrawCurBox 0`" #将旧的方块抹去

((boxCurY = y))

s="$s`DrawCurBox 1`" #显示新的下落后方块

echo -ne $s

else

#走到这儿, 如果不能下落了

Box2Map #将当前移动中的方块贴到背景方块中

RandomBox #产生新的方块

fi

}

#左移一列

function BoxLeft()

{

local x s

((x = boxCurX - 1))

if BoxMove $boxCurY $x

then

s=`DrawCurBox 0`

((boxCurX = x))

s=$s`DrawCurBox 1`

echo -ne $s

fi

}

#右移一列

function BoxRight()

{

local x s

((x = boxCurX + 1))

if BoxMove $boxCurY $x

then

s=`DrawCurBox 0`

((boxCurX = x))

s=$s`DrawCurBox 1`

echo -ne $s

fi

}

#下落到底

function BoxAllDown()

{

local k j i x y iDown s

iDown=$iTrayHeight

#计算一共需要下落多少行

for ((j = 0; j < 8; j += 2))

do

((i = j + 1))

((y = ${boxCur[$j]} + boxCurY))

((x = ${boxCur[$i]} + boxCurX))

for ((k = y + 1; k < iTrayHeight; k++))

do

((i = k \* iTrayWidth + x))

if (( ${iMap[$i]} != -1)); then break; fi

done

((k -= y + 1))

if (( $iDown > $k )); then iDown=$k; fi

done

s=`DrawCurBox 0` #将旧的方块抹去

((boxCurY += iDown))

s=$s`DrawCurBox 1` #显示新的下落后的方块

echo -ne $s

Box2Map #将当前移动中的方块贴到背景方块中

RandomBox #产生新的方块

}

#旋转方块

function BoxRotate()

{

local iCount iTestRotate boxTest j i s

iCount=${countBox[$iBoxCurType]} #当前的方块经旋转可以产生的样式的数目

#计算旋转后的新的样式

((iTestRotate = iBoxCurRotate + 1))

if ((iTestRotate >= iCount))

then

((iTestRotate = 0))

fi

#更新到新的样式, 保存老的样式(但不显示)

for ((j = 0, i = (${offsetBox[$iBoxCurType]} + $iTestRotate) \* 8; j < 8; j++, i++))

do

boxTest[$j]=${boxCur[$j]}

boxCur[$j]=${box[$i]}

done

if BoxMove $boxCurY $boxCurX #测试旋转后是否有空间放的下

then

#抹去旧的方块

for ((j = 0; j < 8; j++))

do

boxCur[$j]=${boxTest[$j]}

done

s=`DrawCurBox 0`

#画上新的方块

for ((j = 0, i = (${offsetBox[$iBoxCurType]} + $iTestRotate) \* 8; j < 8; j++, i++))

do

boxCur[$j]=${box[$i]}

done

s=$s`DrawCurBox 1`

echo -ne $s

iBoxCurRotate=$iTestRotate

else

#不能旋转，还是继续使用老的样式

for ((j = 0; j < 8; j++))

do

boxCur[$j]=${boxTest[$j]}

done

fi

}

#DrawCurBox(bDraw), 绘制当前移动中的方块, bDraw为1, 画上, bDraw为0, 抹去方块。

function DrawCurBox()

{

local i j t bDraw sBox s

bDraw=$1

s=""

if (( bDraw == 0 ))

then

sBox="\040\040"

else

sBox="[]"

s=$s"\033[1m\033[7m\033[3${cBoxCur}m\033[4${cBoxCur}m"

fi

for ((j = 0; j < 8; j += 2))

do

((i = iTrayTop + 1 + ${boxCur[$j]} + boxCurY))

((t = iTrayLeft + 1 + 2 \* (boxCurX + ${boxCur[$j + 1]})))

#\033[y;xH, 光标到(x, y)处

s=$s"\033[${i};${t}H${sBox}"

done

s=$s"\033[0m"

echo -n $s

}

#更新新的方块

function RandomBox()

{

local i j t

#更新当前移动的方块

iBoxCurType=${iBoxNewType}

iBoxCurRotate=${iBoxNewRotate}

cBoxCur=${cBoxNew}

for ((j = 0; j < ${#boxNew[@]}; j++))

do

boxCur[$j]=${boxNew[$j]}

done

#显示当前移动的方块

if (( ${#boxCur[@]} == 8 ))

then

#计算当前方块该从顶端哪一行"冒"出来

for ((j = 0, t = 4; j < 8; j += 2))

do

if ((${boxCur[$j]} < t)); then t=${boxCur[$j]}; fi

done

((boxCurY = -t))

for ((j = 1, i = -4, t = 20; j < 8; j += 2))

do

if ((${boxCur[$j]} > i)); then i=${boxCur[$j]}; fi

if ((${boxCur[$j]} < t)); then t=${boxCur[$j]}; fi

done

((boxCurX = (iTrayWidth - 1 - i - t) / 2))

#显示当前移动的方块

echo -ne `DrawCurBox 1`

#如果方块一出来就没处放，Game over!

if ! BoxMove $boxCurY $boxCurX

then

kill -$sigExit ${PPID}

ShowExit

fi

fi

#清除右边预显示的方块

for ((j = 0; j < 4; j++))

do

((i = iTop + 1 + j))

((t = iLeft + 2 \* iTrayWidth + 7))

echo -ne "\033[${i};${t}H "

done

#随机产生新的方块

((iBoxNewType = RANDOM % ${#offsetBox[@]}))

((iBoxNewRotate = RANDOM % ${countBox[$iBoxNewType]}))

for ((j = 0, i = (${offsetBox[$iBoxNewType]} + $iBoxNewRotate) \* 8; j < 8; j++, i++))

do

boxNew[$j]=${box[$i]};

done

((cBoxNew = ${colorTable[RANDOM % ${#colorTable[@]}]}))

#显示右边预显示的方块

echo -ne "\033[1m\033[7m\033[3${cBoxNew}m\033[4${cBoxNew}m"

for ((j = 0; j < 8; j += 2))

do

((i = iTop + 1 + ${boxNew[$j]}))

((t = iLeft + 2 \* iTrayWidth + 7 + 2 \* ${boxNew[$j + 1]}))

echo -ne "\033[${i};${t}H[]"

done

echo -ne "\033[0m"

}

#初始绘制

function InitDraw()

{

clear

RandomBox #随机产生方块，这时右边预显示窗口中有方快了

RandomBox #再随机产生方块，右边预显示窗口中的方块被更新，原先的方块将开始下落

local i t1 t2 t3

#显示边框

echo -ne "\033[1m"

echo -ne "\033[3${cBorder}m\033[4${cBorder}m"

((t2 = iLeft + 1))

((t3 = iLeft + iTrayWidth \* 2 + 3))

for ((i = 0; i < iTrayHeight; i++))

do

((t1 = i + iTop + 2))

echo -ne "\033[${t1};${t2}H||"

echo -ne "\033[${t1};${t3}H||"

done

((t2 = iTop + iTrayHeight + 2))

for ((i = 0; i < iTrayWidth + 2; i++))

do

((t1 = i \* 2 + iLeft + 1))

echo -ne "\033[${iTrayTop};${t1}H=="

echo -ne "\033[${t2};${t1}H=="

done

echo -ne "\033[0m"

#显示"Score"和"Level"字样

echo -ne "\033[1m"

((t1 = iLeft + iTrayWidth \* 2 + 7))

((t2 = iTop + 10))

echo -ne "\033[3${cScore}m\033[${t2};${t1}HScore"

((t2 = iTop + 11))

echo -ne "\033[3${cScoreValue}m\033[${t2};${t1}H${iScore}"

((t2 = iTop + 13))

echo -ne "\033[3${cScore}m\033[${t2};${t1}HLevel"

((t2 = iTop + 14))

echo -ne "\033[3${cScoreValue}m\033[${t2};${t1}H${iLevel}"

echo -ne "\033[0m"

}

#退出时显示GameOVer!

function ShowExit()

{

local y

((y = iTrayHeight + iTrayTop + 3))

echo -e "\033[${y};0HGameOver!\033[0m"

exit

}

#显示用法.

function Usage

{

cat << EOF

Usage: $APP\_NAME

Start tetris game.

-h, --help display this help and exit

--version output version information and exit

EOF

}

#游戏主程序在这儿开始.

if [[ "$1" == "-h" || "$1" == "--help" ]]; then

Usage

elif [[ "$1" == "--version" ]]; then

echo "$APP\_NAME $APP\_VERSION"

elif [[ "$1" == "--show" ]]; then

#当发现具有参数--show时，运行显示函数

RunAsDisplayer

else

bash $0 --show& #以参数--show将本程序再运行一遍

RunAsKeyReceiver $! #以上一行产生的进程的进程号作为参数

fi