THE UNIVERSITY OF ALABAMA AT BIRMINGHAM

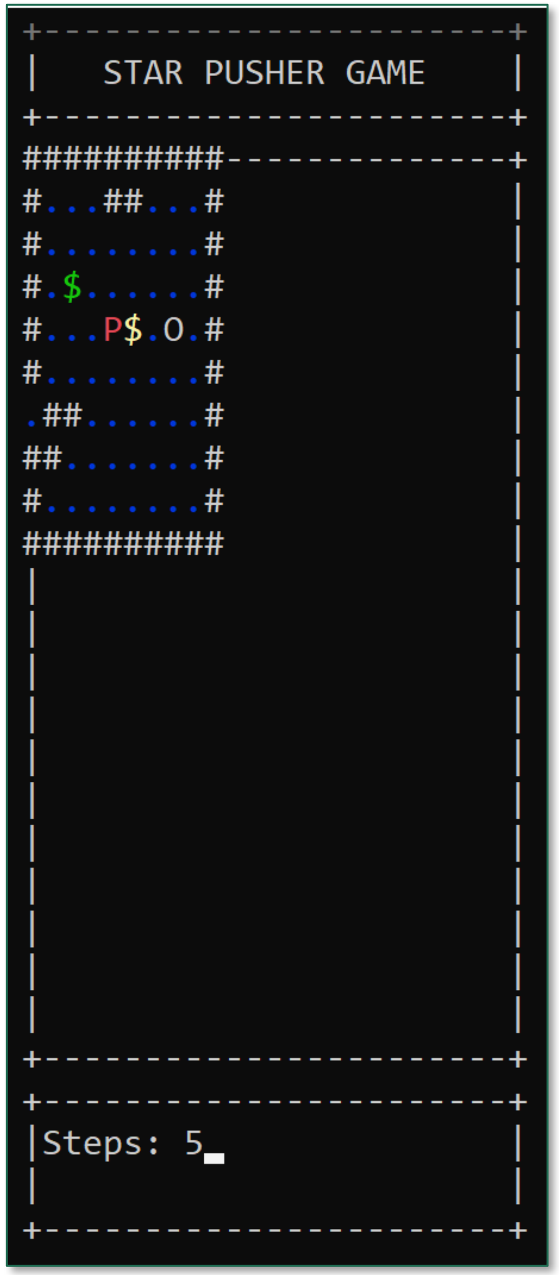
CS330
Sokoban

Spring 2024

C Homework

Goal

- Make a Sokoban-like game
  - (similar to Star Pusher in python)
- Written in C
- Limited by graphics on Vulcan
  - ncurses (ascii-based graphics)



```
+------------------------+
|    STAR PUSHER GAME    |
+------------------------+
##########---------------+
#...##...#               |
#........#               |
#.$......#               |
#...P$.O.#               |
#........#               |
.##......#               |
##.......#               |
#........#               |
##########               |
|                        |
|                        |
|                        |
|                        |
|                        |
|                        |
|                        |
|                        |
|                        |
|                        |
|                        |
+------------------------+
+------------------------+
|Steps: 5                |
|                        |
+------------------------+
```

Game Rules

- Game play occurs on a 2D map, enclosed with walls (no escape)
- Player can move only in cardinal directions: Up, Right, Down, Left
- Player can push star (box) in those same directions
  - Player cannot pull star
- Player cannot move into wall, push star into wall, or push star into another star
- Goal is to push all stars onto a goal square (only one star per goal square) in smallest number of steps (moves)

For an overview of Sokoban, see this link: https://en.wikipedia.org/wiki/Sokoban

A free online version of the game can be found here: https://www.mathsisfun.com/games/sokoban.html

The Game in Code

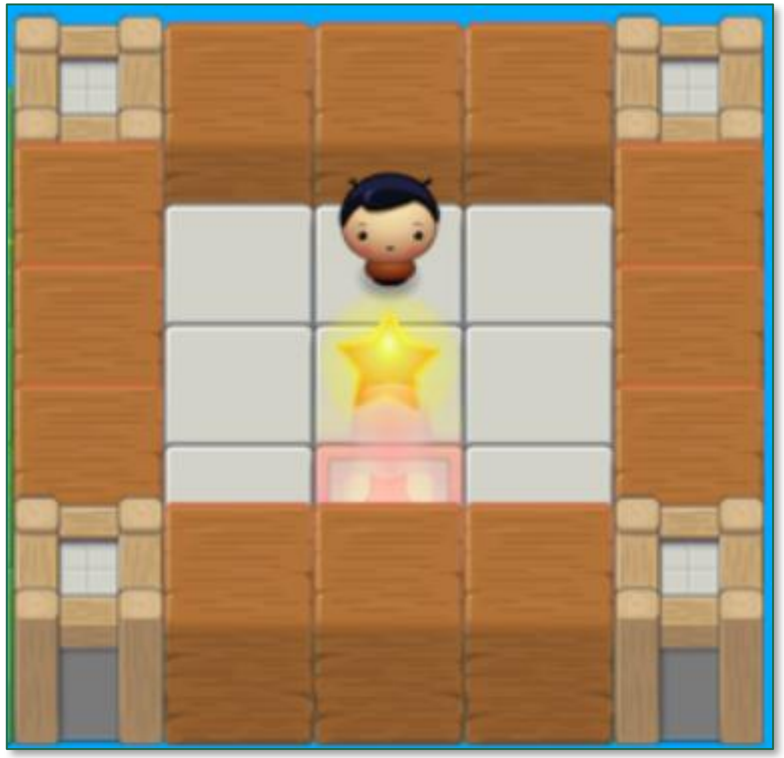
- We’ll need to define some functions:
  - drawMap()
  - validMove() ← you’ll write these
  - movePlayer() ← you’ll write these
- main(){

```
main(){
   // set-up ncurses, variables, load map(s)

   game loop{
      drawMap()
      get_user_input()
      if validMove(){
          movePlayer()
      }
      if playerWon{
          break out of loop and exit
      } // else continue
   }
}
```

Representing/Modelling the Map

- Map gameboard as 2D array
```
int firstMap[5*5] = {
1,1,1,1,1,
1,0,2,0,1,
1,0,3,0,1,
1,0,4,0,1,
1,1,1,1,1};
```
- Where:
  - 0 is blank
  - 1 is wall
  - 2 is Player
  - 3 is Star
  - 4 is Goal square
  - 5 is Star on Goal
  - 6 is Player on Goal
- Also, we'll create some constants to make our life easier (we can use these in our 'for' loops):
  - int MAP_COLS = 10; // number of columns in our map
  - int MAP_ROWS = 10; // number of rows in our map



Representing the Map (cont'd)

- Since a 2D array is really just a 1D array in memory:

```
int firstMap[5*5] = {
    1,1,1,1,1,
    1,0,2,0,1,
    1,0,3,0,1,
    1,0,4,0,1,
    1,1,1,1,1};
```

X

Y

(2,1) (pX ,pY)

*(firstMap+2)

*((firstMap + pY * MAP_COLS) + pX)

== {1,1,1,1,1, 1,0,2,0,1, 1,0,3,0,1, 1,0,4,0,1, 1,1,1,1,1}

- How do we reference a particular element in this 2D array (using the map pointer)?

Modelling a Player as a struct

- What do we need to know about a Player?
  - Current x location
  - Current y location
  - previousSquareValue (was the square the player is on a Goal square?, we need to restore this if the player moves)
- We should place this is a structure
  - Call the x-value: ‘x’
  - Call the y-value: ‘y’
  - Call the previous Square value: ‘$\mathrm{prevSquareValue}$’
- Everything is an int

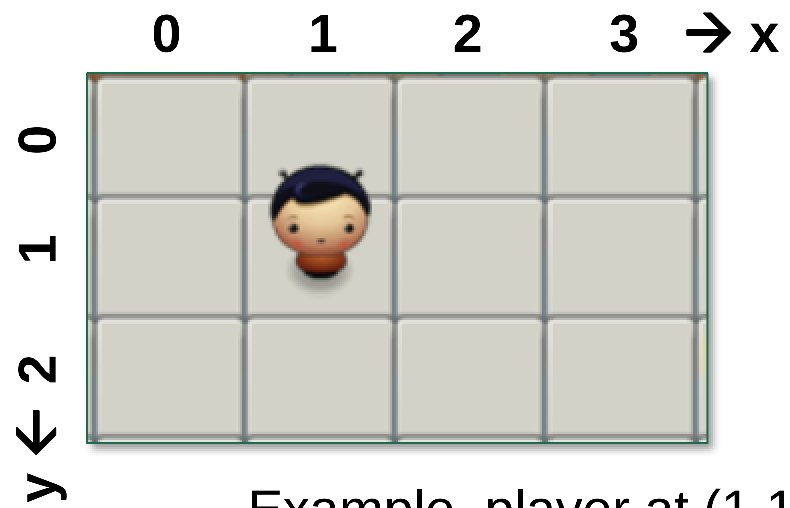
Potential Moves

- Two arrays to quickly obtain new move locations, each element represents: Up, Right, Down, Left (moves clockwise starting with Up)

$\text{int } \mathbf{dX}[4] = \{\mathbf{0,\ 1,\ 0,\ \text{-}1}\};$

$\text{int } \mathbf{dY}[4] = \{\mathbf{\text{-}1,\ 0,\ 1,\ 0}\};$

Up:	delta = 0
Right:	delta = 1
Down:	delta = 2
Left:	delta = 3

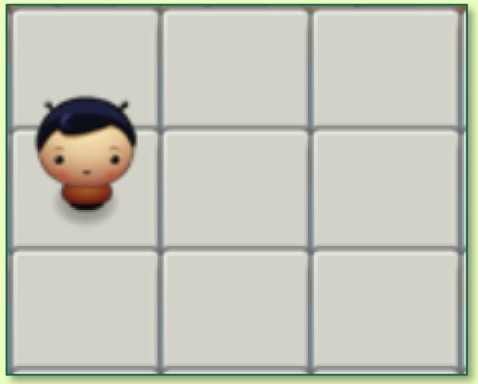
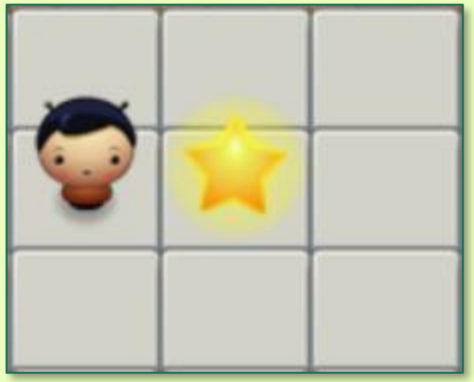
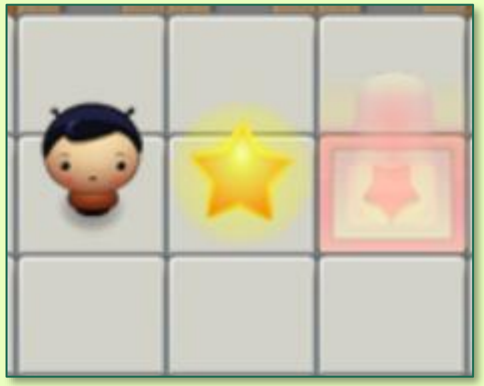
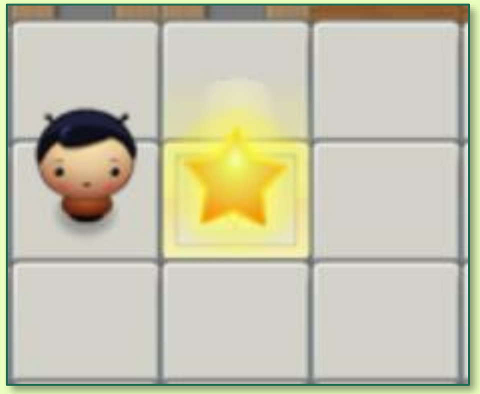
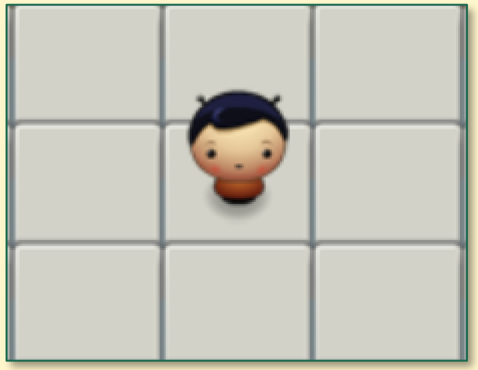
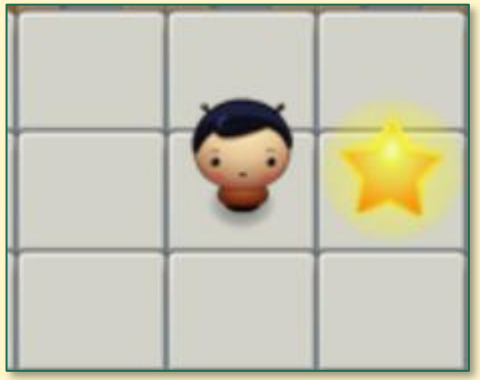
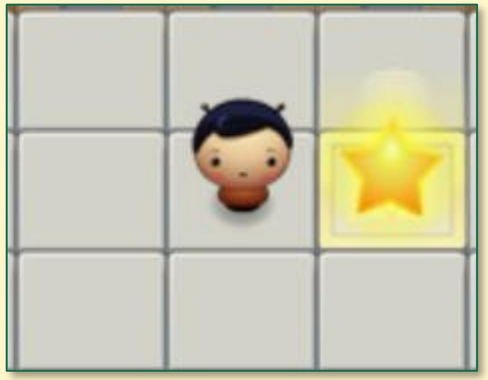
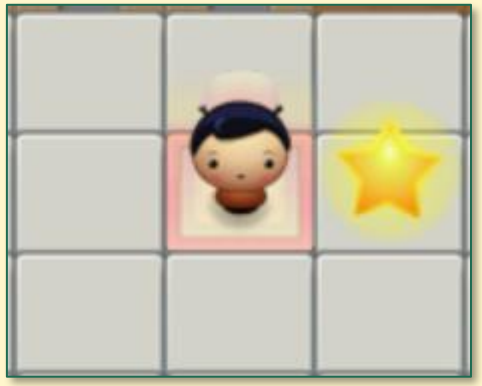


$$\text{int newPlayerX} = \text{p-}{>}\text{x} + \text{dX[delta]};$$
$$\text{int newPlayerY} = \text{p-}{>}\text{y} + \text{dY[delta]};$$

Example, player at (1,1), moves Up:

P->x = 1	newPlayerX = 1 + dX[0] = 1 + 0 = 1
P->y = 1	newPlayerY = 1 + dY[0] = 1 + -1 = 0
delta = 0	player now at (1,0)

Potential moves (cont’d) (moving Right for simplicity, but applicable to all directions)

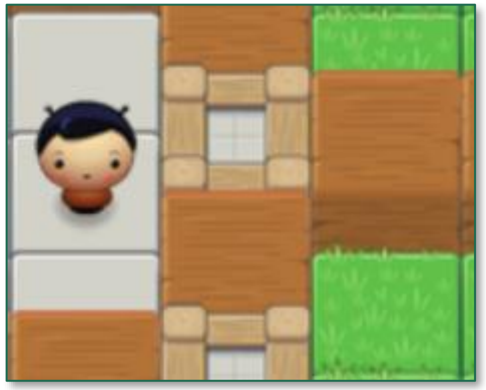
	Empty Square	Star to Empty	Star onto Goal	Star off Goal
Start				
End				

Need to check square we’re moving to, and if it’s a star, also check the square beyond that square

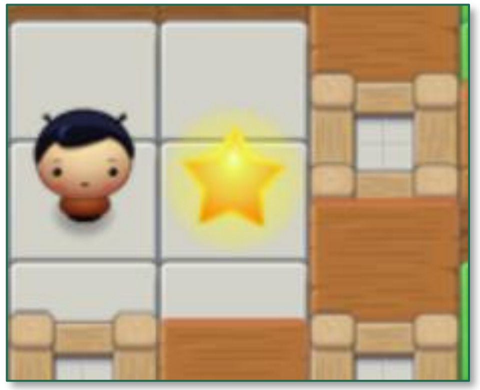
Think about how we need to adjust the Map model to represent these game states

Invalid Moves (when moving Right)

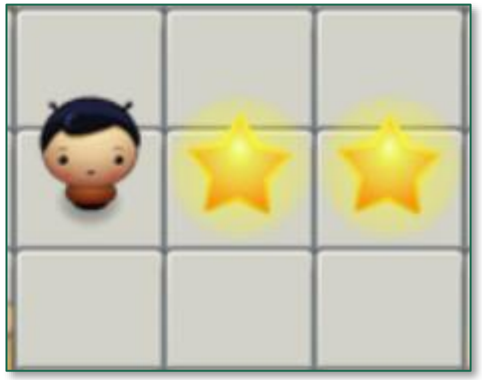
Move
into Wall



Push
Star into Wall



Push
Star into Star



To Begin

- Move the stub code software to Vulcan
  - Either download and save sokoban.zip file to Vulcan
  - Or clone repository on Vulcan
  - Instructions in CS330_C_Bonus_Sokoban.pdf
- Be sure to ‘make’ and ‘make run’ the software to ensure you have all the stub code
  - cs330_sokoban_game.c ← modify this file
  - Makefile
  - maps.txt (this is the map read into the code, in case you want to modify the map)
  - sok_header.h (header info, including Player struct)
  - libsok_helper_vulcan.a (static library with helper functions)

Additional References

- For more on ncurses:
  https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/intro.html

- Decent book, Making Games with Python & Pygame:
  https://inventwithpython.com/pygame/
  The images in this presentation were taken from Sweigart’s Star Pusher game

- Sokoban Map Levels:
  https://inventwithpython.com/starPusherLevels.txt
  http://sneezingtiger.com/sokoban/levels.html
  http://sokobano.de/wiki/index.php?title=Level_format (describes map level format)