Summer 2016

Board Game Application

Final Project – Milestone 2

Problem Description

Your job for this project is to write a board game program called "Checkout".

Full Game Rules:

- Up to 8 players compete to be the first to obtain $1000
- Game takes place on a square board with 5 tile types
  - Empty Tile – No effect
  - Win Tile – Win a random prize
  - Lose Tile – Lose a random prize
  - Grand Prize Tile – Win a grand prize
  - Checkout Tile – Sells all your prizes for cash
- Players can roll 1 - 3 dice to determine how many tiles they move each turn
- If a player lands on the same tile as another player, that player steals a prize and moves 1 tile back

Project Development Process

Your development work on this project has four milestones and therefore is divided into four deliverables. The approximate schedule for deliverables is as follows

- Milestone 1 Due: Friday July 8th at 11:59PM
- Milestone 2 Due: Friday July 15th at 11:59PM
- Milestone 3 Due: Friday July 22th at 11:59PM
- Milestone 4 Due: Friday July 29st at 11:59PM

Design Requirements

1. You must use **`const`** or **`#define directives`** to declare constants. Here is a sample list of constants:

```
MAX_INVENTORY_SIZE 4
TAX .13
```

2. You are allowed to code additional functions.
3. You must document (i.e. comments and indentation) your code properly.
4. You must put the following statement (as a comment) at the beginning of your source code:

```
I declare that the attached assignment is wholly my own work in accordance with
Seneca Academic Policy.  No part of this assignment has been copied manually or
electronically from any other source (including web sites) or distributed to
other students.

Name   _____________________   Student ID  _________________
```

5. You must use blank lines to separate logical units in the source code.
6. You are not allowed to declare global variables.

Milestone #2

Learning Outcomes

Upon successful completion of this milestone, you will be able to:

- Use library functions to validate user input and generate random numbers.
- Use arrays to represent multiple, related pieces of information
- Write functions that accept addresses (pointers) to complete specific tasks.
- Write functions that accept Arrays to complete specific tasks.

In this milestone you will be coding the functions required to make the game playable for 1 player at a time (like the popular Solitaire or Minesweeper games) with simplified rules. For this version the threshold for winning by checking out prizes is lowered to $200 (to make testing easier). This version tracks the player's high score, provides a main menu, and allows the player to choose a one character long name. This version validates all user input and should be able to handle bad input without crashing or getting stuck in an infinite loop.

The **`int main(void)`** function allocates memory to store the player's score, player position, player name, player inventory (can hold up to 10 prizes), the current number of prizes being held by the player, and board size. The main function also stores the high score, and the name of the player who earned the high score in case the user wants to display it. Initialize the high score to 100 and leave the player's name blank. The main function displays a menu where the user can enter:
'p': to (p)lay the game
'q': to (q)uit back to the terminal shell prompt
'r': to show inst(r)uctions
's': to display high (s)core and the name of the player that earned the high score
If the user makes any other choice, inform them that it's an invalid choice and keep trying until the user enters one of p q r s

When the user chooses 'p' to play the game, initialize the player's information by calling the initPlayer function described below (this will among other things ask the user to enter a single character name). Then prompt the user to enter a board size; acceptable board sizes are between 5 and 15 inclusive. Then start the game by calling the playGame function described below. If the player's score is greater than the current high score, replace the high score with the player's score, and save the player's name as the high scoring player name.

When your game is complete you will use the **`srand`** function to make the die rolls unpredictable, but for now do not use **`srand`** so that the outcome of the game is easier to test!

In addition to main() Your milestone must have functions with the following prototypes:

```
int getValidInteger(int low, int high);
```

This function returns an int between the indicated low and high range. If the user does not enter a valid int between low and high (inclusive) display an error message, and try again until the user enters a valid int between low and high

`char getValidCharacter(char low, char high);`

This function returns a char between the indicated low and high range. If the user does not enter a valid char between low and high (inclusive) display an error message, and try again until the user enters a valid char between low and high

`void initPlayer(int* playerScore, int playerPrizes[], unsigned int * prizeCount, char * playerName, int* playerPosition)`

This function will set the value of playerScore to zero, all elements of playerPrizes to 0, the value of prizeCount to 0, the value of playerPosition to 0, and it will prompt the user to enter a one character long name and store it in the value of playerName.

`void playGame(unsigned int boardSize, int *playerScore, int playerPrizes[], unsigned int* prizeCount, char * playerName, int* playerPosition);`

Until the player wins, repeat the following steps:

- Display the board including the player's position on the board
- Display the player's score, and inventory information
- Let the player roll 1-3 dice.
- Advance the player by the number of tiles indicated by the dice
- Depending on the tile that the player lands on, something special may happen:
    - 'G' – player wins a grand prize (call the winGrandPrize function)
    - 'W' – player wins a regular prize (call the winPrize function)
    - 'L' – player loses a random prize (call the losePrize function)
    - 'C' – the player converts the accumulated prizes (call the checkout function) to cash and adds to their score. If the player's score is now over the threshold ($200 in this version), then the player wins the game!

`char getDisplayTile(unsigned int index, unsigned int playerPosition, char playerName);`

This function works similarly to the getTileType function from Milestone 1, except in the case that index is equal to playerPosition, in which case this function returns playerName instead.

`void displayBoard(unsigned int size, unsigned int playerPosition, char playerName);`

Replace your existing displayBoard function with the upgraded version above. This new version works as the previous version, except that it draws the playerName in the board position indicated by playerPosition so that the player can tell where they are on the board.

`int getRandom(int low, int high);`

Returns a random integer between low and high inclusive.

`unsigned int playerRoll();`

Prompt the user to enter a number of dice to roll between 1 and 3 inclusive. Simulate the number of die rolls indicated by the user, displaying each number rolled, and return the sum of the rolls.

`void winPrize(int playerPrizes[], unsigned int* prizeCount);`

Generate a new prize by sampling a random number between 10 and 100 inclusive. If the value of prizeCount is less than the maximum number of prizes that the player can hold at a time (10) then store the new prize in playerPrizes and update the value of prizeCount. Indicate to the user the value of the prize won, or a message indicating that the player's inventory is full.

`void winGrandPrize(int playerPrizes[], unsigned int* prizeCount);`

Same as winPrize, except the value of the grand prize is between 100 and 200 inclusive.

`int loseItem(int playerPrizes[], unsigned int* prizeCount);`

If the player has any prizes, choose one at random, remove it, and reduce the value of prizeCount as appropriate. Notice you cannot simply set the value of playerPrizes at some index to 0, you have to think about this.

`int checkout(int* playerScore, int playerPrizes[], unsigned int* prizeCount);`

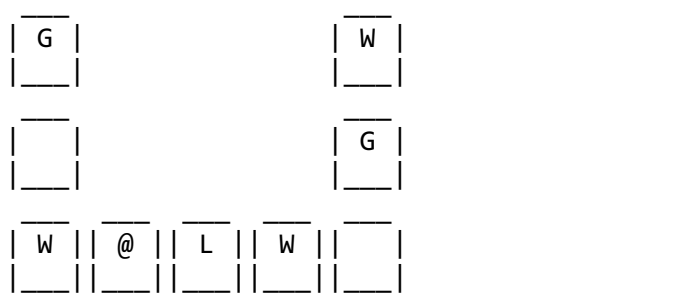
Add the values of all prizes the player holds to the value of playerScore. Set the count of prizes back to 0. Display the amount checked out (won) by the player. Return true if the value of playerScore is now greater than or equal to the winning threshold (200), and return false otherwise.

See the following page for program completion and submission:

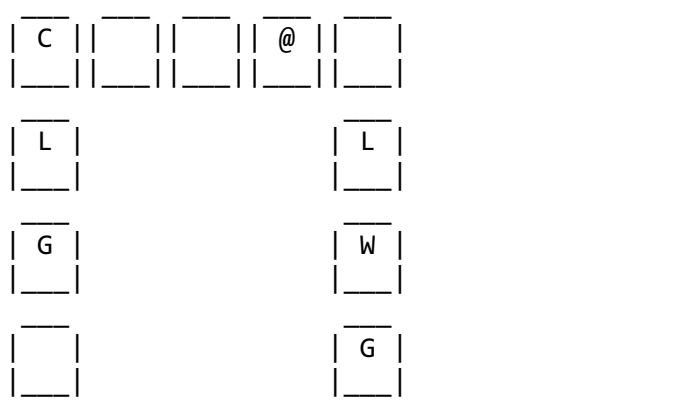
Program completion

Your program is complete if your output matches the following output when the **srand** function is not used or commented out. Green numbers show the user's input.

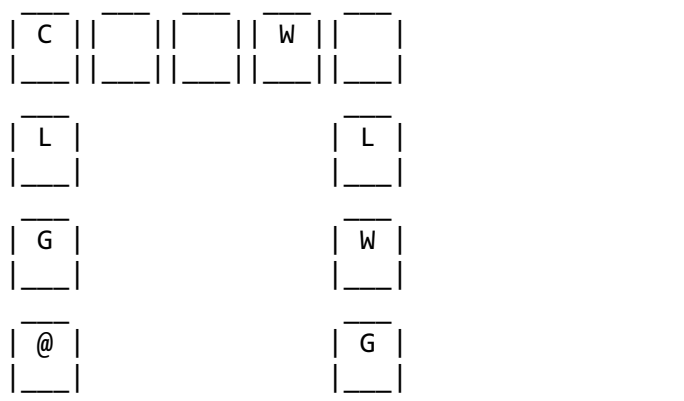
```
Welcome to CHECKOUT
Main Menu

p-(p)lay q-(q)uit r-inst(r)uctions s-HI(s)core:
p
number of players is 1
Enter player ID: @
Enter board size: 5
 ___  ___  ___  ___  ___
| @ ||   ||   || W ||   |
|___||___||___||___||___|
 ___                 ___
| L |               | L |
|___|               |___|
 ___                 ___
| G |               | W |
|___|               |___|
 ___                 ___
|   |               | G |
|___|               |___|
 ___  ___  ___  ___  ___
| W ||   || L || W ||   |
|___||___||___||___||___|
score: 0   inventory (0 items):
your turn, how many dice will you roll?  4
Invalid input, try again: 0
Invalid input, try again: 9
Invalid input, try again: lol
Invalid input, try again: 3
You rolled: 2 5 4
Advancing 11 spaces
nothing happens, go again.
 ___  ___  ___  ___  ___
| C ||   ||   || W ||   |
|___||___||___||___||___|
 ___                 ___
| L |               | L |
|___|               |___|
```



score: 0   inventory (0 items):
your turn, how many dice will you roll? 2
You rolled: 2 6
Advancing 8 spaces
you won a prize of 76

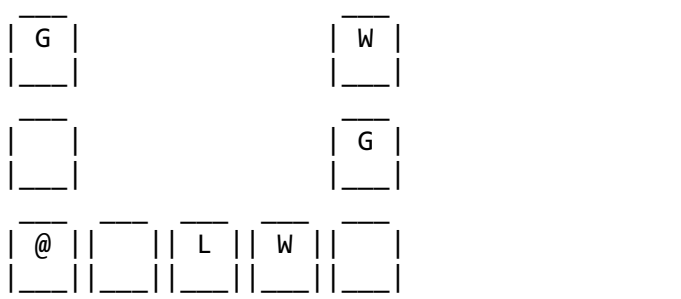


score: 0   inventory (1 items): 76,
your turn, how many dice will you roll? 3
You rolled: 5 1 4
Advancing 10 spaces
nothing happens, go again.

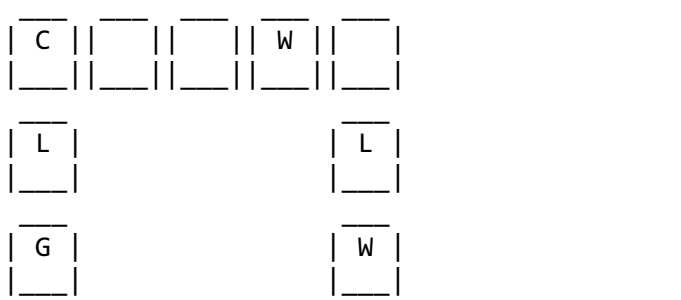


score: 0   inventory (1 items): 76,

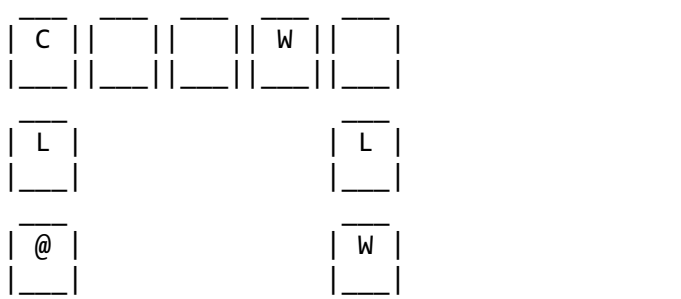
```
your turn, how many dice will you roll?  1
You rolled: 2
Advancing 2 spaces
Player lost a prize valued at 76!
 ___  ___  ___  ___  ___
| C ||   ||   || W ||   |
|___||___||___||___||___|

 ___                 ___
| @ |               | L |
|___|               |___|

 ___                 ___
| G |               | W |
|___|               |___|

 ___                 ___
|   |               | G |
|___|               |___|

 ___  ___  ___  ___  ___
| W ||   || L || W ||   |
|___||___||___||___||___|
score: 0   inventory (0 items):
your turn, how many dice will you roll?  1
You rolled: 2
Advancing 2 spaces
nothing happens, go again.
 ___  ___  ___  ___  ___
| C || @ ||   || W ||   |
|___||___||___||___||___|

 ___                 ___
| L |               | L |
|___|               |___|

 ___                 ___
| G |               | W |
|___|               |___|

 ___                 ___
|   |               | G |
|___|               |___|

 ___  ___  ___  ___  ___
| W ||   || L || W ||   |
|___||___||___||___||___|
score: 0   inventory (0 items):
your turn, how many dice will you roll?  3
You rolled: 3 2 6
Advancing 11 spaces
you won a prize of 24
 ___  ___  ___  ___  ___
| C ||   ||   || W ||   |
|___||___||___||___||___|

 ___                 ___
| L |               | L |
|___|               |___|
```



score: 0   inventory (1 items): 24,
your turn, how many dice will you roll? 1
You rolled: 1
Advancing 1 spaces
nothing happens, go again.

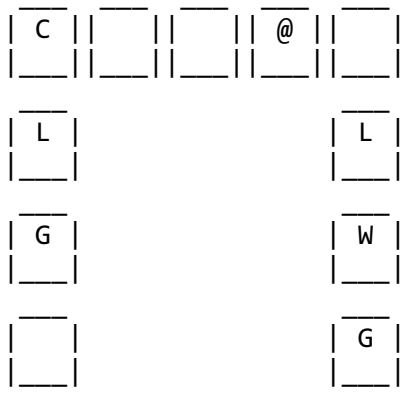


score: 0   inventory (1 items): 24,
your turn, how many dice will you roll? 1
You rolled: 1
Advancing 1 spaces
you won a grand prize of 189

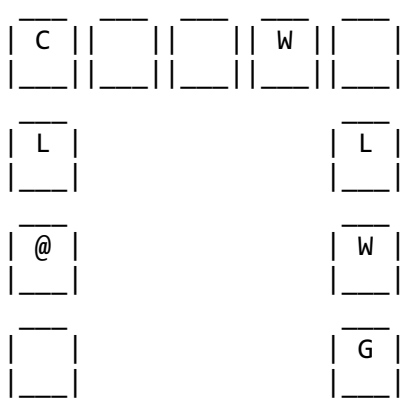


score: 0   inventory (2 items): 24, 189,

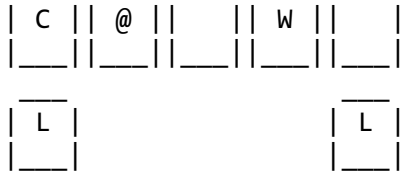
```
your turn, how many dice will you roll?  1
You rolled: 5
Advancing 5 spaces
you won a prize of 69
```

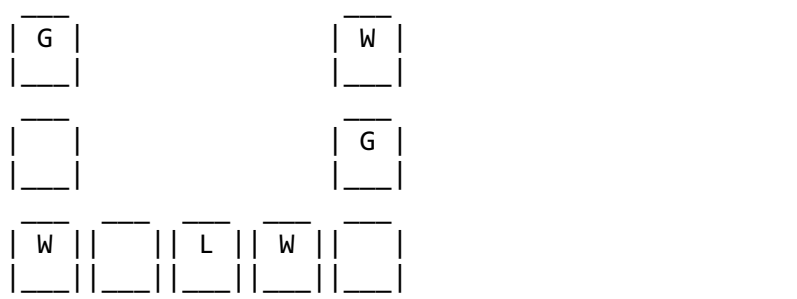


```
score: 0   inventory (3 items): 24, 189, 69,
your turn, how many dice will you roll?  3
You rolled: 3 4 4
Advancing 11 spaces
you won a grand prize of 138
```

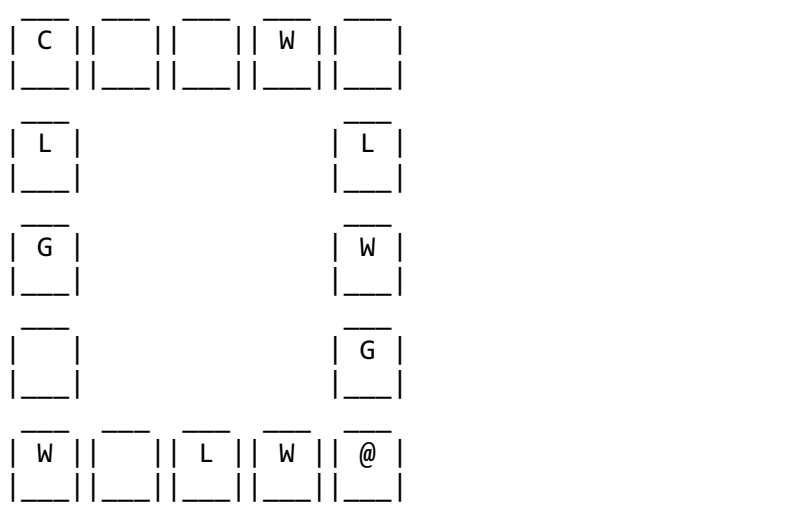


```
score: 0   inventory (4 items): 24, 189, 69, 138,
your turn, how many dice will you roll?  1
You rolled: 3
Advancing 3 spaces
nothing happens, go again.
```

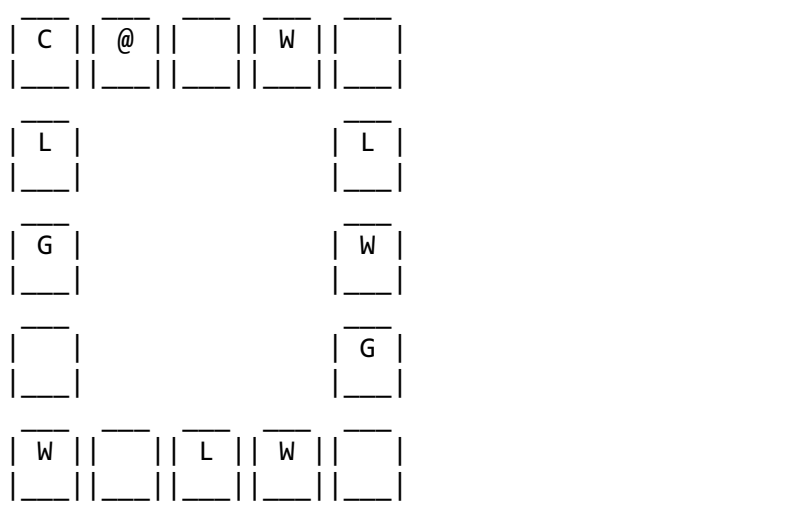




score: 0   inventory (4 items): 24, 189, 69, 138,
your turn, how many dice will you roll? 3
You rolled: 3 2 2
Advancing 7 spaces
nothing happens, go again.



score: 0   inventory (4 items): 24, 189, 69, 138,
your turn, how many dice will you roll? 3
You rolled: 2 6 1
Advancing 9 spaces
nothing happens, go again.



score: 0   inventory (4 items): 24, 189, 69, 138,

```
your turn, how many dice will you roll?  3
You rolled: 1 1 6
Advancing 8 spaces
you won a prize of 40
 ___  ___  ___  ___  ___
| C ||   ||   || W ||   |
|___||___||___||___||___|

 ___                 ___
| L |               | L |
|___|               |___|

 ___                 ___
| G |               | W |
|___|               |___|

 ___                 ___
|   |               | G |
|___|               |___|

 ___  ___  ___  ___  ___
| W ||   || L || @ ||   |
|___||___||___||___||___|
score: 0   inventory (5 items): 24, 189, 69, 138, 40,
your turn, how many dice will you roll?  2
You rolled: 2 5
Advancing 7 spaces
You checkedout for $460.00 score is now: $460.00
You won the game!
Main Menu

p-(p)lay q-(q)uit r-inst(r)uctions s-HI(s)core:
s
__
  \_______
   \++++++|
    \=====|
    0---  0

HI SCORE: 460 Player Name: @
Main Menu

p-(p)lay q-(q)uit r-inst(r)uctions s-HI(s)core:
q
This game is much more fun than bash...
```