Machine Project I: **Ghostless Pacman**

Content

INTRODUCTION:
who is Pacman and why are there no ghosts?

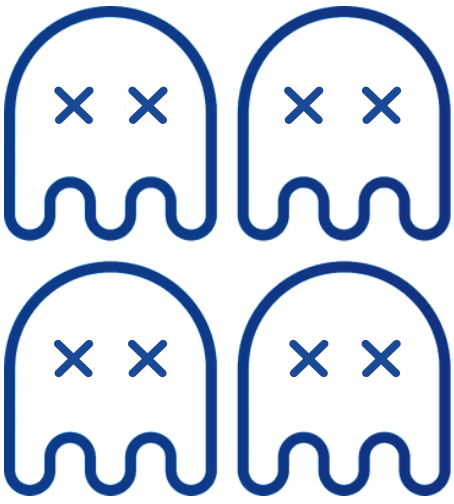
Ghostless Pacman is a 2D game made with C
and a simplified version of the famous
maze video game, Pac-Man.

In this post-apocalyptic world where ghosts are no longer living,
the player controls a character named Pacman.
He is alone. Nothing to run from. No ghosts to play with.

He realized that he is more frightened in solitary than a ghost's company.

Suddenly, the Lord of Console had approached him and
gave him a chance to revive the ghosts.

Since then, Pacman dedicated his life to this endeavor.



GAME MANUAL:
how to be pro at this

When the program is started, you will encounter the main lobby that contains options to play, to learn, and to exit.

After choosing the option to play, you will have to choose the number of food you want to eat in the game.

Subsequently, you will spawn in a 10x10 world that holds all the element you need to revive the ghosts.

HINT: Press S if you want to generate a new world before playing ;)

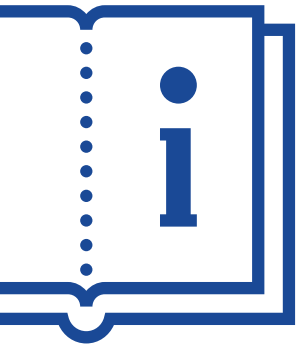
The game is simply played by pressing W, A, S, and D, or even its lowercase. Although it kinda sucks, you have to press enter every move.

The main objective of this game is to first collect ALL THE FOOD symbolized by '@', and then to exit the world by going to the symbol '**X**'. Be careful while navigating your Pacman as it would die when it hits a block, symbolized by '**W**', or the border of the world.

While playing or after you win or lose, you can press M to return to main menu.

That's it! You are now ready to be a pro player in Ghostless Pacman.

If you want more details, you can always access the instructions inside the program.



DATA STRUCTURES

variables

1. **Constants**
   - Board dimension
     - **#Define boardDim 12**
       Sets the size of the board's array.
   - Content dimension
     - **#Define randDim 10:**
       Sets the size of the board's content's array.
2. **Arrays and Coordinates**
   - Game board
     - *char* **boardArray[boardDim][boardDim]**
       Contains the content of the game.
   - Content coordinates
     - *int* **blockRow[randDim], blockCol[randDim]**
       Contains the X, Y coordinates of the blocks.
     - *int* **foodRow[randDim], foodCol[randDim]**
       Contains the X, Y coordinates of the food.
     - *int* **exitRow, exitCol**
       Contains the X, Y coordinates of the exit.
3. **User-dependent Game Contents**
   - Pacman
     - *int* **PacX, PacY**
       Contains X, Y coordinates of Pacman.
   - Food
     - *int* **foodChoice**
       Stores the player's choice of food frequency.
     - *int* **foodCounter**
       Counts the number of food eaten by Pacman.

user-defined functions

1. **Content Randomization**
   - Array shuffler
     - *void* **shuffle(int *a, int *b)**
       Shuffles the passed array's content with no repetition.
   - Block randomizer
     - *void* **blockRandomize(int blockRow[randDim], int blockCol[randDim])**
       Randomizes the coordinates of the block.
   - Food randomizer
     - *void* **foodRandomize(int foodRow[randDim], int foodCol[randDim])**
       Randomizes the coordinates of the food.

DATA STRUCTURES

user-defined functions

2. **Game Board Structures and Contents**
   - Border filler
     - *void* **borderFill**(**char boardArray**[**boardDim**][**boardDim**])
       Places border around the game board.
   - Block filler
     - *void* **rand_blockFill**(**char boardArray**[**boardDim**][**boardDim**], **int exitRow**, **int exitCol**, **int blockRow**[**randDim**], **int blockCol**[**randDim**])
       Fills the game board with blocks randomly.
   - Food filler
     - *void* **rand_foodFill**(**char boardArray**[**boardDim**][**boardDim**], **int blockRow**[**randDim**], **int blockCol**[**randDim**], **int foodRow**[**randDim**], **int foodCol**[**randDim**], **int foodChoice**)
       Fills the game board with food randomly.
   - Initial board printer
     - *void* **print_initialBoard**(***char boardArray***[**boardDim**][**boardDim**])
       Prints out the board while the player prepares for his battle stance.
   - Current round's board printer
     - *void* **printBoard**(**char boardArray**[**boardDim**][**boardDim**], **int foodChoice**, **int foodCounter**)
       Prints out the board for the current round.
   - Board updater
     - *void* **roundUpdate**(***char boardArray***[**boardDim**][**boardDim**], ***int PacY***, ***int PacX***)
       Updates the board for every move of Pacman.
3. **Game Mechanics**
   - Win checker
     - *void* **checkWin**(**int foodChoice**, **int foodCounter**)
       Checks if the player has collected the right amount of food before exiting the game board.
   - Player wins
     - void **playerLose**()
       Displays a message in the console when the player has lost.
   - Player loses
     - void **playerWin**()
       Displays a message in the console when the player has won.

DATA STRUCTURES

standard library functions

1. **Random Number Generator**
   - Including <stdlib.h>
     - `rand()`
       Returns a pseudo-random number from 0 to RAND_MAX, a big constant that varies.
     - `srand()`
       Sets the starting point for producing a series of pseudo-random integers called a seed.
   - Including <stdlib.h> and <time.h>
     - `srand(time(NULL))`
       Makes use of computer's internal clock; hence, the seed always changes.
2. **Console Clearer**
   - Including <stdlib.h>
     - `system("clear")`
       Clears the previous output in the console.

THE ALGORITHM

algorithm 1: main lobby

```
1  Print Welcome Sign
2  Print Main Menu
3  Input a number from the player, save as menuChoice
4      If menuChoice == 1,  then
5          Input a number from the player, save as foodChoice
6              If foodChoice >= 2 and <= 9, then
7                  Print initial game board
8                  Do Algorithm 2
9                  Return to line 1
10             Else
11                 Return to line 5
12             End if
13     Else if menuChoice == 2, then
14         Print Instruction Menu
15         Input a letter from the player, save as instrucChoice
16             If instrucChoice == 'A' or 'a', then
17                 Print Instruction 1
18             Else if instrucChoice == 'B' or 'b', then
19                 Print Instruction 2
20             Else if instrucChoice == 'C' or 'c', then
21                 Print instruction 3
22             Else if instrucChoice == 'D' or 'd', then
23                 Print instruction 4
24             Else if instrucChoice == 'E' or 'e', then
25                 Return to line 1
26             Else
27                 Print Invalid
28                 Return to line 15
29             End if
30     Else if menuChoice == 3, then
31         End program
32     Else
33         Print Invalid
34         Return to line 1
35     End if
```

THE ALGORITHM

algorithm 2: board initialization

```
1  Initialize the randomization of blocks, foods, and exit{
2      Randomize 10 unique X and Y coordinates of blocks
3      Randomize foodChoice unique X and Y coordinates of food from
4          Algorithm 1
5      Declare exitCol ← Generate a random number from 0 to 9
6      Declare exitRow ← Generate a random number from 0 to 9
7  }
8
9  Initialize the game board{
10     Construct the board borders
11     Fill the starting position (1, 1) with Pacman
12     Fill the board randomly with an exit using exitCol and
13         exitRow
14     Fill the board randomly with 10 blocks such that it is
15         still winnable
16     Fill the board randomly with foodChoice food taken from
17         Algorithm 1
18 }
19 Print game board
20 Input a letter from the player, save as pressChoice
21     If pressChoice == 'P' or 'p', then
22         Do Algorithm 3
23         Return to line 1 in Algorithm 1
24     Else if pressChoice == 'S' or 's', then
25         Return to line 9
26     Else
27         Return to line 20
28     End if
```

THE ALGORITHM

algorithm 3: game start

```
1  Initialize game statistics{
2      Declare PacX ← 1
3      Declare PacY ← 1
4      Declare foodCounter ← 0
5  }
6
7  Start loop
8      Setup the game{
9          Set the game conditions{
10             If Pacman collides with the border or with the
11                 blocks, then
12             Print Player Loses
13             Lock the user controls
14             Input a letter from the player, save as resultChoice
15                 If resultChoice == 'M' or 'm', then
16                     Go to line 70
17                 Else
18                     Return to line 14
19                 End if
20             Else if Pacman eats a food, then
21                 Add 100 to foodCounter
22             Else if Pacman goes to the exit, then
23                 If all food are eaten, then
24                     Print Player Wins
25                     Lock the user controls
26                     Input a letter from the player, save as
27                         resultChoice
28                         If resultChoice == 'M' or 'm', then
29                             Go to line 70
30                         Else
31                             Return to line 26
32                         End if
33                 Else
34                     Print Player Loses
35                     Lock the user controls
36                     Input a letter from the player, save as
37                         resultChoice
```

THE ALGORITHM

algorithm 3: game start

```
38                      If resultChoice == 'M' or 'm', then
39                          Go to line 70
40                      Else
41                          Return to line 36
42                      End if
43                  End if
44              End if
45          }
46          Update the game board
47          Print the game board
48      }
49
50      Input a letter from the user, save as playChoice
51          If playchoice == 'W' or 'w', then
52              Move Pacman upward
53              Print blank on Pacman's previous location
54              Return to line 7
55          Else if playchoice == 'A' or 'a', then
56              Move Pacman leftward
57              Print blank on Pacman's previous location
58              Return to line 7
59          Else if playchoice == 'S' or 's', then
60              Move Pacman downward
61              Print blank on Pacman's previous location
62              Return to line 7
63          Else if playchoice == 'D' or 'd', then
64              Move Pacman rightward
65              Print blank on Pacman's previous location
66              Return to line 7
67          Else
68              Print Invalid
69          End if
70  End loop
```

ERROR HANDLING

content randomization

Because the numbers generated to place exit, food, blocks are random, we have to make sure that they meet the standards of the game.

In this notion of randomness, there comes a possibility of the contents' coordinates to repeat or to coincide with each other. Therefore, certain measures and conditions were set so that it would be prevented. Take a look at figures 1 and 2 to know how these problems were solved.

Figure 1. An implementation of an algorithm that uses functions to randomize coordinates without repetition.

```
8  void shuffle(int *a, int *b){
9      // Stores the value of the array of its current index
10     // into temporary storage, ie blockStore
11     int blockStore = *a;
12     // Shuffles the array
13     *a = *b;
14     // Ensures that there are no repeating numbers
15     *b = blockStore;
16 }
17
18 void blockRandomize(int blockRow[randDim], int blockCol[randDim]){
19
20     int blockCount, blockNum, blockNum2;
21
22     // Block randomizer
23     for (blockCount = 0; blockCount < randDim; blockCount++){
24         // This generates an array, i.e. blockRow, of numbers
25         // containing from 1 to 10
26         blockRow[blockCount] = blockCount + 1;
27     }
28     for (blockCount = 0; blockCount < randDim; blockCount++){
29         // Assigns blockNum a random number from 1 to 10
30         blockNum = (rand()%9)+1;
31         // Shuffles the array in a way that there is no repetition
32         shuffle(&blockRow[blockCount], &blockRow[blockNum]);
33     }
34     // Generating another unique shuffled array to be used for the column
35     for (blockCount = 0; blockCount < randDim; blockCount++){
36         blockCol[blockCount] = blockCount + 1;
37     }
38     for (blockCount = 0; blockCount < randDim; blockCount++){
39         blockNum2 = (rand()%9)+1;
40         shuffle(&blockCol[blockCount], &blockCol[blockNum2]);
41     }
42 }
```

ERROR HANDLING

content randomization

Figure 2. An implementation of algorithms that uses functions to control the spawn of the blocks and food.

```
83  void rand_blockFill(char boardArray[boardDim][boardDim], int exitRow, int exitCol,
84                      int blockRow[randDim], int blockCol[randDim]){
85
86      // Placing blocks
87      for (int countRow = 0; countRow < boardDim; countRow++){
88          for(int countColumn = 0; countColumn < boardDim; countColumn++){
89              for (int arrayCount = 0; arrayCount < randDim; arrayCount++){
90                  // Ensures that the blocks won't block Pacman and the exit
91                  while ((boardArray[blockRow[arrayCount]][blockCol[arrayCount]] == 'X') ||
92                  (blockRow[arrayCount] == (exitRow + 1) && blockCol[arrayCount] == exitCol) ||
93                  (blockRow[arrayCount] == (exitRow - 1) && blockCol[arrayCount] == exitCol) ||
94                  (blockRow[arrayCount] == exitRow && blockCol[arrayCount] == (exitCol + 1)) ||
95                  (blockRow[arrayCount] == exitRow && blockCol[arrayCount] == (exitCol - 1)) ||
96                  (boardArray[blockRow[arrayCount]][blockCol[arrayCount]] == '<') ||
97                  (blockRow[arrayCount] == 1 && blockCol[arrayCount] == 1) ||
98                  (blockRow[arrayCount] == 2 && blockCol[arrayCount] == 1) ||
99                  (blockRow[arrayCount] == 2 && blockCol[arrayCount] == 2)){
100                     blockRandomize(blockRow, blockCol);
101                 }
102                 boardArray[blockRow[arrayCount]][blockCol[arrayCount]]='W';
103             }
104         }
105     }
106 }
```

```
108 void rand_foodFill(char boardArray[boardDim][boardDim], int blockRow[randDim],
109                    int blockCol[randDim], int foodRow[randDim], int foodCol[randDim], int foodChoice){
110
111     for (int countRow = 0; countRow < boardDim; countRow++){
112         for(int countColumn = 0; countColumn < boardDim; countColumn++){
113             for (int arrayCount = 0; arrayCount < foodChoice; arrayCount++){
114                 // Ensures that the food won't be blocked by the blocks and won't concide with Pacman and the exit.
115                 while
116                 ((boardArray[foodRow[arrayCount]][foodCol[arrayCount]] == 'W') ||
117                 (foodRow[arrayCount] == (blockRow[arrayCount] + 1) && foodCol[arrayCount] == blockCol[arrayCount]) ||
118                 (foodRow[arrayCount] == (blockRow[arrayCount] - 1) && foodCol[arrayCount] == blockCol[arrayCount]) ||
119                 (foodRow[arrayCount] == blockRow[arrayCount] && foodCol[arrayCount] == (blockCol[arrayCount] + 1)) ||
120                 (foodRow[arrayCount] == blockRow[arrayCount] && foodCol[arrayCount] == (blockCol[arrayCount] - 1)) ||
121                 (boardArray[foodRow[arrayCount]][foodCol[arrayCount]] == '<') ||
122                 (boardArray[foodRow[arrayCount]][foodCol[arrayCount]] == 'X')){
123                     foodRandomize(foodRow, foodCol);
124                 }
125                 boardArray[foodRow[arrayCount]][foodCol[arrayCount]]='@';
126             }
127         }
128     }
129 }
```

ERROR HANDLING

content randomization

Although we already have set conditions to control this dilemma, we have to really make sure that if it ever happens, i.e. winning is impossible, there is a way to fix it. See figure 3 to see how this was implemented.

Figure 3. The implementation of this algorithm gives the player a chance to generate another world to play on by pressing S before the game starts.

```
printf("Hint: Stuck? Press S to generate a new world!\n");
scanf(" %c", &pressChoice);
if (pressChoice == 'p' || pressChoice == 'P'){
    system("clear");
    pressLoop++;
}
// If bad RNG happens, there should be a salvation key xD
else if (pressChoice == 's' || pressChoice == 'S'){
    system("clear");
    blockRandomize(blockRow, blockCol);
    foodRandomize(foodRow, foodCol);
    exitRow = (rand()%9)+1;
    exitCol = (rand()%9)+1;
}
else {
    noreturnLoop++;
    system("clear");
    fgetc(stdin);
}
```

...and for those naughty players

In every game, there will always be a player who inputs what is not given; even though it is explicitly asked for. Or to give them the benefit of the doubt, maybe they did it accidentally.

To solve this, we want to use **switch statements** to print Invalid whenever an unexpected input is given to the program. Also, we made use of the function **fgetc(stdin)** to make sure that a **scanf()** loop is stopped when it happens in this event.

EASTER EGG!

a game would be too boring without one, wouldn't it?

You want a hint?

This is your hint

You want another hint?

He who created sea.