Santorini - 230 ver.

Project Profile

The goal of this project is to implement the game Santorini using the C programming language. The game will be interactive and played between a human user and an AI (artificial intelligence). The focus of this project is primarily on exercising an introductory understanding of the C programming language including basic data types, looping and conditional constructs, arrays, iteration, basic I/O, formatted output, and functions. This semester, all projects are individual work (not done in groups), unlike the challenges and labs!

Santorini

Traditionally, Santorini game is a 2-4 player game in which the players take turns moving one of his/her builders into an octagonal neighbor space, then construct a building level adjacent to the builder. Building level is from 2 to 4 and the objective of the game is to become the first player who moves a builder into a level 3 building to win the game. In your implementation you will use P (for Player) and A (for AI) characters in place of the colors.

In this project, we made a 230-ver for this game so the rules will be different.

1. you only need to implement this game for **two players** and each only has **1 builder** to play in this game. Please use 'P' as player's builder and 'A' to represent the AI's builder.
2. Both builders can be moved into any adjacent space no matter what building levels are.
3. A builder cannot be moved into any adjacent space if the space is occupied by another builder.
4. When you move your builder into a space, your builder constructs a building level for each space (excludes the space the builder just moved in) that is located in a straight line of any octagonal direction if it is not **blocked** by another player.
5. When AI moves its builder into a space, AI's builder destructs a building level for each space (excludes the space the builder just moved in) that is located in a straight line of any octagonal direction if it is not **blocked** by another player.
6. The building level ranges from 0 to 4. Your game should not contain any building with a building level more than 4 or less than 0 during the game play.
7. The game ends if there are at least 10 spaces with a building level of 4 or at least 10 spaces with a building level of 0 in the game board.
8. The human player wins if there are at least 10 spaces with a building level of 4 where the AI player wins if there are at least 10 spaces with a building level of 0 when the game ends.

Game Board

The game board is a 6-column by 6-row suspended grid. The rows are labeled 1 - 6 and the columns are labeled 1 - 6. This is the initial game board:

	1	2	3	4	5	6
1	2	2	2	2	2	2
2	2	2	2	2	2	2
3	2	2	2	2	2	2
4	2	2	2	2	2	2
5	2	2	2	2	2	2
6	2	2	2	2	2	2

The rows are labeled from top to bottom starting with 1, and the columns are labeled from left to right starting with 1. The number in the table(i, j) means the point of octagon at row i and column j.

Game Play

Your implementation will prompt the user before game play to determine where (which octagon) player's builder is. After the player chooses a starting point, AI chooses a space adjacent to the player's starting point as its starting point. Notice that choosing starting points won't change any building level in the game board. For example, Here the game states if the human player chooses (1,2) as the starting point, then AI chooses (1,3) as its starting point.

	1	2	3	4	5	6
1	2	P	A	2	2	2
2	2	2	2	2	2	2
3	2	2	2	2	2	2
4	2	2	2	2	2	2
5	2	2	2	2	2	2
6	2	2	2	2	2	2

In your implementation, have the human player take the first move. When it is the human's turn the game will prompt the player for which (row, column) she would like to move in the game board. For example, using the above game state, if the human player chooses (2,3) the resulting game state would be:

	1	2	3	4	5	6
1	2	**3**	A	**3**	2	2
2	**3**	**3**	**P**	**3**	**3**	**3**
3	2	**3**	**3**	**3**	2	2
4	**3**	2	**3**	2	**3**	2
5	2	2	**3**	2	2	**3**
6	2	2	**3**	2	2	2

In each turn, both Player and AI need to move their builder to an octagon adjacent to the builder's current location. Take the below game board for example, in player's turn, she can move her builder into (1, 2), (1, 4), (2, 2), (2, 4), (3, 2), (3, 3) or (3, 4).

```
    1 2 3 4 5 6
1   2 3 A 3 2 2
2   3 3 P 3 3 3
3   2 3 3 3 2 2
4   3 2 3 2 3 2
5   2 2 3 2 2 3
6   2 2 3 2 2 2
```

Notice that the player cannot choose (1, 3) since the octagon(1, 3) is occupied by AI's builder. The player cannot choose octagon(2, 3) since her builder needs to be moved. The player cannot choose octagon(5, 6) since octagon(5, 6) is not adjacent to octagon(1, 2). You need to tell the player if she chooses an invalid move, like (1, 3) or (5, 6) in this situation, and ask the player to choose another move.

After moving a builder each turn, the building level for each space is updated based on where the builder moves into. If a space is located in a straight line of any of the builder's octagonal direction, and there's no other builder on the line, then it's building level gets updated. The building level goes up or down depends on who controls the builder. In Player's turn, the building level goes up where it goes down in AI's turn. Here is an example, the player moves its builder into (1, 2), it levels up all space heighted by yellow color as below. (1, 3), (1, 4), (1, 5), and (1, 6) is not updated because the paths to these spots are blocked by AI's builder, which is located in (1, 3). **Also, the level of the octagon that builders move onto does not increase/decrease.** In this example, level of (1, 2) will keep the same.

```
    1 2 3 4 5 6
1   2 P A 3 2 2
2   3 3 3 3 3 3
3   2 3 3 3 2 2
4   3 2 3 2 3 2
5   2 2 3 2 2 3
6   2 2 3 2 2 2
```

After the end of Player's turn, the next move would be decided by the AI player, followed by prompting the human player again. Game play would then proceed in that fashion. Here is an example of the game playing:

```
    1 2 3 4 5 6
1   2 3 1 2 1 2
2   3 3 P A 2 2
3   2 3 2 2 1 2
4   3 1 3 1 3 1
5   1 2 3 1 2 3
6   2 2 3 1 2 2
```

```
    1 2 3 4 5 6
1   2 4 1 3 1 2
2   4 4 4 A 2 2
3   3 P 3 3 2 3
4   4 2 4 1 3 1
5   1 3 3 2 2 3
6   2 3 3 1 3 2
```

```
    1 2 3 4 5 6
1   2 3 0 2 1 2
2   3 3 A 1 1 1
3   3 P 2 2 2 3
4   4 2 3 1 2 1
5   1 3 2 2 2 2
6   2 3 2 1 3 2
```

```
    1 2 3 4 5 6
1   3 3 0 2 2 2
2   3 4 A 2 1 1
3   4 4 P 3 3 4
4   4 3 4 2 2 1
5   2 3 3 2 3 2
6   2 3 3 1 3 3
```

```
    1 2 3 4 5 6
1   3 3 0 1 1 2
2   2 3 3 A 0 0
3   4 4 P 2 2 4
4   4 3 4 1 2 0
5   2 3 3 1 3 2
6   2 3 3 0 3 3
```

The game ends if there are at least 10 spaces with a building level of 4 or at least 10 spaces with a building level of 0 in the game board. The human player wins if there are at least 10 spaces with a building level of 4 where the AI player wins if there are at least 10 spaces with a building level of 0 when the game ends. **When the game ends, output a message to announce the winner.** For example, output "**Player wins!**".

Requirements

Your primary objective is to use the C programming language to design and implement a Santorini game that operates according to the game play described in the previous section. You are required to design and implement the appropriate data structures and corresponding algorithms that will enable a human player to play against an AI player.

The AI design is entirely up to you, however, it should be at least as smart as a 5 year old playing the game. That is, the AI should make some obvious choices such as **moving into an achievable octagon so it can decrease at least one building level from 4 to 3 or 1 to 0 if possible**. If you have taken a course in Artificial Intelligence you are welcome to make your AI more sophisticated (e.g., Minimax), however, this is not required. Furthermore, you are constrained to using only the C constructs that we have covered up through the first week of class. In particular:

Allowable C Language Constructs	
● printf ● basic C data types and variables (int, float, double, char, _Byte/byte) ● storage sizes and ranges ● type specifiers ● arithmetic expressions ● for, while, do loop, if statement, switch, conditional operator ● aligning output ● scanf	● 1D array and initialization ● const ● multi-dim arrays ● variable length arrays ● array length bounds, iteration, length ● functions, arguments, locals, returns ● prototype declaration ● functions and arrays, mutability ● global variables ● automatic/static variables

In addition, your implementation must meet the following specific requirements:

1. An array must be used to represent the game board as described above.
2. Must be able to take "row column" as user input (i.e. "2 4" to play the tile in the second row and fourth column).
3. One or more functions must be used as part of the implementation.
4. Arrays must be passed to functions as arguments.
5. Iteration must be used to traverse the game board.
6. Your implementation must be able to determine end state of game: win or lose.
7. An AI must exist in your game as described above.
8. You should minimize the use of global variables - no global variables is the best.
9. Your implementation must show the state of the game, modeled after our examples above, after each move (both human and AI) and display the final game state, who won or who drew. All output should be properly formatted and aligned.

Video Demonstration

You must provide a link to a 2-minute video demonstration of your game being played. Your video should be short and get to the point, showing a game being played to completion. That is, displaying each alternating game state, final state, and who won (or if it was a draw). You must provide a voice in the video demonstration (silence is not allowed); explain what you are doing when you play the game. **We only accept a link of a video, e.g., a YouTube video. Uploading a video on Gradescope will result in 2 in the video category**.

Grading and Rubric

You will be graded according to the following rubric. The scoring of the rubric below falls under various categories and headings. Categories may also be multiplied - that is, some categories count more than others (e.g, x2, x3).

1. Delivery (40 points)
    a. Implements the appropriate data structures and corresponding algorithms that will enable a human player to play against an AI player with the game flow described above.
    b. Must be able to take “row column” as user input.
    c. Must handle edge cases and invalid inputs properly, including reprompting the user as necessary. You can expect the inputs are always two numbers in the following format: “row column”.
    d. Your implementation must be able to determine the end state of the game: win or lose.
    e. An AI must exist in your game as described above.
    f. Your implementation must show the state of the game, modeled after our examples above, after each move (both human and AI) and display the final game state, who won or who drew, along with the recorded points in the form mentioned in the previous section. All output should be properly formatted and aligned.
2. Design (30 points)
    a. Functions are declared and used properly. One or more functions must be used as part of the implementation.
    b. Arrays must be passed to functions as arguments.
    c. Iteration must be used to traverse the game board.
    d. Data structures and data types are declared and used properly.
    e. Variable and function naming is clear and help with understanding the purpose of the program.
    f. Global variables are minimized, declared, and used properly.
    g. Control flow (e.g., if statements, looping constructs, function calls) are used properly.
    h. Algorithms are clearly constructed and are efficient. For example, there is no extra looping, unreachable code, confusing or misleading constructions, and missing or incomplete cases.
    i. No use of disallowed C constructs such as structs or pointers.
3. Coding Style (5 points)
    a. Code is written in a consistent style. For example, curly brace placement is the same across all if/then/else and looping constructors.
    b. Proper and consistent indenting is adhered to across the entire implementation.
    c. Proper spacing is used making the code understandable and readable.
4. Comments (5 points)
    a. Comments in the code are used to document algorithms.
    b. Variables are documented such that they aid the reader in understanding your code.
    c. Functions are documented to indicate the purpose of the parameters and return values.
5. Video (14 points)
    a. The video shows the code being compiled from the command line.

   b. The video shows the code being executed and satisfying the output requirements.
   c. We can reproduce the game shown in your video by running your code
6. README.txt (5 points)
   a. The README.txt file is well written.
   b. The README.txt file provides an overview of your implementation.
   c. The README.txt file explains how your code satisfies each of the requirements.
7. GradeScope Submission (1 point)
   a. You automatically get a point for submitting to gradescope.

Submission

You must submit two files to Gradescope by the assigned due date:

- **`Santorini.c`** - this is your implementation of the game.
- **`README.txt`** - this is a text file containing a brief 1 paragraph overview of your submission highlighting the important parts of your implementation. The goal should make it easy and obvious for the person grading your submission to find the important rubric items. This text file should also clearly include a URL to your video for us to review. Make sure the video is not private or otherwise inaccessible to the graders.

Make sure you submit this project to the correct assignment on Gradescope! Remember, projects are to be done individually, not in groups!.

You are required to make your program compilable and runnable in the VM environment we give you.