Group Project - Instructions

- Add your group members' names to the top of EACH code file in the comment provided.

- Do not communicate with other groups about the project. This will be considered cheating. If you have general questions about using Visual Studio or XCode use the discussion forum on Canvas.

- Read through the existing code. Make note of which variables, constants, and functions are defined where.

  Character symbols used in the game:

`~` tilde	`^` caret	`<` left angle bracket	`.` period
`@` "at" sign	`&` ampersand	`>` right angle bracket	`O` letter oh
`#` pound sign	`=` equal sign	`:` colon	

- Look at the keyboard constants listed in interactionFunctions.cpp. Run the already-compiled game to experiment with the keyboard instructions and see what they're supposed to do.

- *Follow the instructions carefully.* **Follow the instructions carefully.** Follow the instructions carefully. I cannot stress enough the importance of attention to detail. Don't overthink the problem, just follow directions.

- Do not alter function names or parameters unless specifically instructed to do so. Do not alter any code that is specifically marked "DO NOT CHANGE OR REMOVE".

- When writing code, use existing functions instead of rewriting the logic. Yes, this matters to your grade.

1. In all files, remove all unnecessary function prototypes. This will make things easier for you going forward. Remember: a function prototype is unnecessary if the function is called below where it is defined or if it is called in another file. Do not skip this step.

2. In utilityFunctions.cpp edit the function `abs(...)` so that it returns the absolute value of its input.

3. In utilityFunctions.cpp edit the function `sign(...)` so that it returns the sign of its input. This function returns 0 if the input is 0, 1 if the input is positive, or -1 if the input is negative.

4. In werewolfFunctions.cpp the function `werewolfIsAlive()` should return `true` only if the werewolf has health greater than zero.

5. In werewolfFunctions.cpp the function `werewolfIsStunned()` should return `true` only if the werewolf's stun count is greater than zero.

6. In werewolfFunctions.cpp the function `getWerewolfSymbol()` should return a different symbol depending on whether the werewolf is alive but not stunned, alive and stunned, or dead.

7. In <u>werewolfFunctions.cpp</u> the function `doWerewolfHit(...)`
    a. Subtract `hitpoints` from `werewolfHealth`.
    b. Make sure `werewolfHealth` is never below zero.
    c. Add 2 to `werewolfStunnedCount`.
    d. return `werewolfHealth`

8. In <u>werewolfFunctions.cpp</u> edit the function `isOpenSpaceForWerewolf(...)` so that the werewolf can also walk over key, pebble, pebbles, plank, plank_set, and rope squares. The werewolf cannot walk over tied ropes.

9. In <u>mapFunctions.cpp</u> fill in the missing code for `getMapSquare(...)` so that it returns the expected value.

10. In <u>mapFunctions.cpp</u> fill in the missing code for `setMapSquare(...)` so that it sets the expected value.

11. In <u>mapFunctions.cpp</u> edit the function `canSeePast(...)` so that it returns `true` if the square is empty or has a chasm, key, pebble, pebbles, plank, plank_set, rope, rope_tied, or slingshot.

12. In <u>playerFunctions.cpp</u> in `getLookingAtLocation(...)` :
    a. Edit the last two parameters so they can be used to return data to the calling function.
    b. Add code to the cases inside the `switch` statement to make the function work.

13. In <u>playerFunctions.cpp</u> delete `getLookingAtX(...)` and `getLookingAtY(...)` and update the rest of the code to use `getLookingAtLocation(...)`.

14. In <u>mapFunctions.cpp</u> edit the function `saveGame(...)` so that it works correctly. Make sure it saves all necessary information to the gameSlotN.txt file. Compare it to `loadGame(...)` to check. You should be able to save a game and then immediately load it and see no difference in your gameplay. You *will* need to edit the function signature; add any new parameters after the existing parameters.

15. In <u>mapFunctions.cpp</u> edit the function `loadGame(...)` so that it works correctly. Take note of new information being read from the gameSlotN.txt files.

16. In <u>werewolfFunctions.cpp</u> the function `doWerewolfNextMove(...)`
    a. If the werewolf is not alive, stop executing this function.
    b. If the werewolf's turn should be skipped this round, *reset the flag* and stop executing this function.
    c. If `werewolfStunnedCount` is greater than zero, subtract one from `werewolfStunnedCount` and stop executing this function.
    d. On the line `bool randomlyPickX = (true /* MISSING CODE */);` replace the `true` with code that will randomly return `true` or `false` with equal probability. <u>Make sure your code does not produce any errors nor warnings for this line.</u>

17. In <u>mapFunctions.cpp</u> in the function `getRandomEmptyLocation(...)`
    a. Edit the parameter types to pass by reference the arguments required to make this function work.
    b. Fix the line `const int stopAfterThisMany = 0 /* MISSING CODE */;` so that `stopAfterThisMany` is a random number picked from the possible number of empty locations.

18. In <u>mapFunctions.cpp</u> edit the two marked `if` statements in function `twoLocationsAreVisibleToEachOther(...)` so that it works correctly. Try drawing a couple situations by hand to figure out what the `if` statements are checking.

19. In <u>playerFunctions.cpp</u> fix the function `playerIsLookingAt(...)` to return `true` if the player can see the given location AND the player is looking in the direction of the given location. Both criteria are important.

20. In <u>interactionFunctions.cpp</u> in the function `doUse(...)` inside the case for using the slingshot:
    a. Fix the `if` statement so that the Boolean expression in the `if` statement is `true` if the player is looking at the werewolf <u>AND</u> the werewolf is within shooting range of the slingshot.
    b. Fix the code `int damage = 0 /* MISSING CODE */;` so that `damage` is a value from 1 to `SLINGSHOT_MAX_DAMAGE` (inclusive).

21. In <u>interactionFunctions.cpp</u> in `doCheckForPlayerDamage()` if the werewolf catches the player:
    a. Fix the first `if` statement at the top of the function. There's a hint in the comment. Also, the werewolf must still be alive in order to catch the player.
    b. Fix the code `int damage = 0 /* MISSING CODE */;` so that `damage` is a value between 1 and `WEREWOLF_MAX_DAMAGE` (inclusive).
    c. Fix the second `if` statement in the function. Use a meaningful function call.
    d. Fix the code `string verbToUse = VERBS[0 /* MISSING CODE */];` so that it randomly selects one of `VERBS_COUNT` strings from the array `VERBS`.
    e. Fix the code `lastMessage += " of damage and has been teleported to a random location in the maze.";` so that if the werewolf cannot be teleported elsewhere the player is notified.

22. In <u>interactionFunctions.cpp</u> in `doLoadDefaultMap()` add code to handle werewolf data.

23. In <u>interactionFunctions.cpp</u> in `doCommand(...)`add code to prevent the werewolf from moving when the user types the help, load game, or save game commands.

24. In <u>playerFunctions.cpp</u> add code to the `for` loop in `getFarthestActionableLocation(...)` to make it work. Do not edit the `for` loop definition. Do not edit code outside the `for` loop.

Give yourself enough time to merge everyone’s changes together, and test the resulting code. Make sure it compiles, and test each of the player commands, including whether you can save and load your games.