This worksheet has been developed as part of ongoing research on the instruction of problem solving by Dr. Judi McCuaig (University of Guelph). For more information or to get involved, please contact Hillary Dawkins at hdawkins@uoguelph.ca.

Tic-Tac-Toe Problem-Solving Worksheet

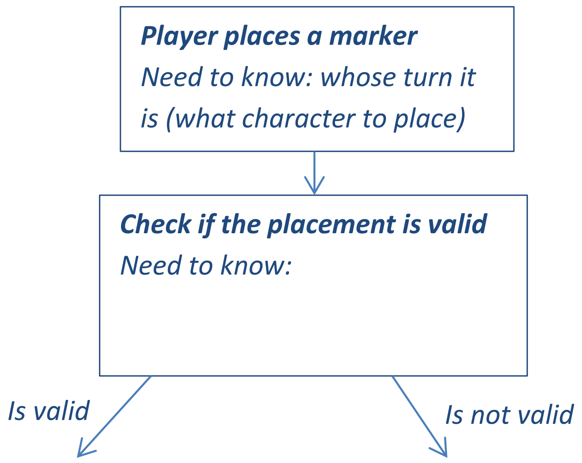
In this worksheet, we will work through the problem solving checklist as applied to the tic-tac-toe task. The relevant sections have been annotated with hints and ***partial*** possible responses to help you get started. Use the blank spaces provided as a rough scratchpad for your thoughts.

1. Understanding the problem	
a) Identify what you don't know	Read the description carefully and highlight terms you are not familiar with, keep a list of questions to ask the instructor or research later
- *I am not sure or can't remember what ncurses is used for* - *Are we assuming 3 by 3 tic-tac-toe, or a general game?*	
b) Research	Look up terms, ask for clarification, answer any questions defined in part 1.a
Confer with your group and facilitator; write the answers to your questions (from above) here. You can come back and add to this section during the group discussion.	
c) Consolidation	Restate the problem in your own words
Try to expand on the problem description; be specific. Hint: do not use the word 'tic-tac-toe'. You might identify more questions at this stage, add them to the list.	
d) Deepening understanding	List potential user stories and possible inputs to the program, write down the expected output in general terms
As a user of this program, I would expect that I can: - *Choose a spot for my marker* - *Place my marker* - *Be informed when the game is over*	

Reconvene with the group to discuss your work

2. Planning

a) Task decomposition	Begin by making a list of all the components you will need to solve the problem. For each component, further break down the task into subtasks (tree structure) so that each subtask does only one simple thing (i.e. it could be contained in a single function).


Player places a marker
Need to know: whose turn it is (what character to place)
Check if the placement is valid
Need to know:
Is valid
Is not valid

List of functions and variables I need:

- *Variable turn: int or boolean to keep track of whose turn it is*
- *Variable gameOn: int or Boolean to keep track of the state of the game*

b) Research	Research the best tools to use for each subtask.
For this task, we have been provided with a library of functions for using ncurses. Let's research these functions by looking at the main.c file in the TTTLibrary. What does 'moveCursor' do and how is it different from 'move'? What input does 'drawCharacter take'? Add your own test cases to this file and check if you understand the usage of all functions provided to you. Use the blank space to record your findings.	
c) Write pseudocode	Based on your research, write a plan of how each subtask will be implemented.

d) Make time estimates	Consider your prior experience with the subtask, whether new tools have to be installed, and the complexity of your pseudocode to assign time estimations to each of your subtasks.

** Reconvene with the group to discuss your work **

3. Implementation

e) Environment	Build your base folder structure and skeleton code, get a simple program to compile and run using your skeleton environment and input data.
Go back to the problem description; do you need to adhere to a certain folder structure? Do you need your program to run on a certain system? Make a note or draw a diagram of your environment here with folders and the files you will need in each folder. *Construct your environment and add minimal code files where appropriate. Construct your makefile and check that you can create your executable as instructed in the problem description.*	
f) Code task-by-task	Now we are ready to begin writing code for each sub-task identified in part 2.a. Test each sub-task as you go (achieving expected outputs for a list of given inputs).
Use the space provided to record any problems that you encounter.	

4. Evaluation

a) Brainstorm test cases	Write down a list of potential test cases; think about possible inputs, especially edge cases. Create or reuse a testing library based on your list.

What do I expect to happen when:

- *A user tries to place a marker where one already exists?*
- *The board is full and no one wins (stalemate)?*

b) Test cases	Implement test cases and make notes of anything that needs to be fixed. Re-enter phase 3 and come back to this step as needed.

Use this space to record any problems that you encounter.

c) Verification	Look back at the problem description and the user stories you identified in part 1.a. Have you met all requirements and satisfied the assignment as a whole?

Make a checklist based on the problem description, your problem restatement, and your user stories:

- *My code compiles without warnings using the appropriate flags*
- *A user can place a marker on any empty spot*
- *A user is informed when the game has ended and is told who won*

d) Reflection	Reflect on your learning. What was the biggest take-away? What are you proud of? What skills will you use again in the future?