C++ 2025B - MTA - Exercises

Requirements and Guidelines

The exercises in the course would require you to implement a Tank Battle game as a console application. The game is somewhat based on an old video game, described here. However you shall follow the instructions as set in this document.

Note: This exercise is purely for educational and programming practice. It does not promote real-world conflict. The tank battle scenario is simply in a game-based context.

Environment and Submission Instructions:

The exercise should be implemented in Visual Studio 2022 or later, with standard C++ libraries and run on Windows with Console screen of standard size (80*25), using gotoxy for printing at a specific location on screen (X, Y), using _kbhit and _getch for getting input from the user without blocking, and Sleep for controlling the pace of the game.
(All the above would be explained to you during one of the next lab sessions).
Submission is in MAMA, as a single zip file containing **only** the code and the vcxproj and sln files + readme.txt with IDs -- but without the DEBUG folder and without any compiled artifact. (Please **make sure to delete** hidden directories and unnecessary binary files).

Note that you MUST provide a readme.txt file that would contain the IDs of the students submitting the exercise. Even if there is only a single submitter, you still need to put your ID in the readme.txt file. Your readme file will also list any additional bonus features you've implemented, if you'd like to request bonus points for them.

Important notes on GenAI and using external code snippets:

- You may use ChatGPT or other GenAI tools, but remember that the responsibility for meeting the requirements and submitting working code is on you. Which means that you have to carefully read and understand any piece of code that you integrate into your submission.
- There are some existing C++ implementations for this project out there on the web. Please note that any implementation that you find would probably not fit exactly the requirements of this exercise. Adapting existing code for the purpose of the exercise would probably be much more complicated than implementing your own code.
- **Important:**
  If you do choose to rely on AI generated code or on existing project that you found on the web, or from any other source, you are required to add comments above any big portion of code that you didn't write and is integrated into your project (for any snippet of 5 lines of code or more). The comment should point at the source of the code (link to the web, a brief summary of the prompt used with the AI tool to get this code, etc.). It would be emphasized again that: (a) it is valid to integrate code that you didn't write! (b) you are still responsible for making sure the code works and fits its purpose. (c) you are required to have a clear comment indicating the origins of this code snippet. (d) slight adaptations to the code do not make it "yours", so even if you rearrange it a bit, change variable names etc., but it is still an external code of 5 or more lines of code, you are still required to add the origins comment as explained above.
- The instructions above are relevant also for integrating pieces of code that you got from the lab exercise or other resources provided by the course staff, etc. You should document that as well with a proper comment, indicating the origins of the code.

Exercise 1
In this exercise you will implement the basic “Tank Battle” game for human players.

You decide how to use the size of the console screen (80 width on 25 height) for:
- Presenting the board.
- Presenting number of points, “remaining lives” information, etc.

Menu
The game shall have the following entry menu:
(1) Start a new game
(8) Present instructions and keys
(9) EXIT

When the game starts, there will be two tanks presented on screen, inside a maze, without any movement. Once the users select move direction (using the keys, as listed below) their tank would move in this direction even if they do not press any key, as long as game board boundaries are not reached, the tank doesn’t hit a wall and the “STAY” key isn’t pressed.
The keys for movement are controlling separately each continuous track (זחל) of the tank.
Input should be case insensitive (you should allow both capital and small letters).

Keys:

	Player 1	Player 2
RIGHT track forward	E	O
RIGHT track backwards	D	L
LEFT track forward	Q	U
LEFT track backwards	A	J
STAY	S	K
Shoot	W	I (i)

When only one track is moving the tank is rotating, according to following rules:

Keys	Movement
RIGHT track forward ONLY	Rotate counter-clockwise in place, including diagonal steps (8 steps to complete 360°).
RIGHT track forward LEFT track backwards	Rotate counter-clockwise in place, without diagonal steps (4 steps to complete 360°).
LEFT track forward ONLY	Rotate clockwise in place, including diagonal steps (8 steps to complete 360°).
LEFT track forward RIGHT track backwards	Rotate clockwise in place, without diagonal steps (4 steps to complete 360°).

Game Elements and Flow
The gameboard is consisted of:
- Two tanks, standing somewhere on screen, one tank per player. Each tank has a cannon, pointing to one of 8 possible directions (U, UR, R, DR, D, DL, L, UL)
- Mines, located in different places on screen
- Walls, creating the maze
- Shells - the tank artillery (cannonball), appears on screen following a tank firing.

Game Element	Suggested Char*
Tank	O (big O)
Cannon UP, DOWN	| above or below the tank
Cannon UP-RIGHT, DOWN-LEFT	/ to the up-right or down-left of the tank
Cannon UP-LEFT, DOWN-RIGHT	\\ to the up-left or down-right of the tank
Cannon LEFT, RIGHT	- (dash)
Mine	@
Wall	#
Tank Shell	*

* Notes:
1. You may use other chars and document it both in your readme file and the game instructions screen, make sure to use reasonable chars.
2. You may use colors, but as explained below you should still allow the game to run in a non-colored mode.

Note that a tank cannot be completely adjacent to a wall, if the cannon is in the direction of the wall. This means that the tank cannot rotate to a direction with a wall blocking its cannon!

When a tank is being shot or steps on a mine the game ends and the other tank is the winner. There can be a tie if the two tanks are dead in the same game-loop cycle. When there is a win or a tie, the game should show a proper message, wait for any key to be pressed and return to the main menu (beware of calling the menu from the game, which may result with a bad recursive flow that goes infinitely down the stack).

After shooting, the tank cannot shoot (pressing the shoot key is being ignored) for 5 game-loop cycles. Tanks can move and shoot in the same game-loop cycle. If a shell hits another shell, both explode. Tanks are dead only if a shell exactly hits the tank or the tank exactly steps over a mine. If a shell hits the cannon of a tank, the cannon will be ruined and disappear but the tank would be still alive, without a cannon. Cannon stepping over a mine has no effect, as the cannon is above ground (the mine would visually disappear when the cannon is stepping over it, but would still be there and would be displayed again when not hidden by the cannon, or stepped on with a tank).

When a shell hits a wall, the wall is weakened, so after two hits the wall will “fall” (disappear). You can present a “weakened” wall (that was hit once) differently, or not.

The boundaries of the game board are considered as an invisible tunnel to the other side of the board (from top to bottom and vice versa and from left to right and vice versa). This applies for both the tanks and the shells (i.e. both tanks and shells may continue their movement to the other side through the invisible tunnel).

Pausing a game

Pressing the ESC key during a game pauses the game. You should present a message on screen saying: “Game paused, press ESC again to continue or X to go back to the main menu”.
When the game is at a pause state, pressing ESC again would continue the game, with all moving game objects (tanks, shells) continuing their movement exactly as it was before pausing, as if the game hadn’t paused. Pressing X or x in pause mode ends the current game returning to the main menu (see comments about avoiding a recursive flow).

Notes to be aware of:

1. Do not use the command ‘exit’:
   Using the command exit for finishing the program is **not allowed**. In general using ‘exit’ is a bad sign. You should finish the program by normally finishing main.
2. Make sure your program does not perform unnecessary bad recursive calls. For example, if your menu calls “game.run()” for starting a game, and then the game calls “menu.run()” for presenting the menu after the game ends, this sounds like a bad recursion that should be avoided.

Bonus Points:

You may be entitled for bonus points for additional features, such as adding colors or other nice features. Note: there will not be any bonus for music or any feature that requires additional binary files to be part of your submission! Adding large required binary files to your submissions may subtract points!

Important notes for getting bonus:

1. If you decided to add colors (as a bonus feature) please add an option in the menu to run your game with or without colors (the default can be to use colors, but the menu shall allow a switch between Colors / No Colors) - to allow proper check of your exercise in case your color selection would not be convenient for our eyes. The game MUST work properly in the No Colors selection.

   Note: If you do want to support colors, use the following way for doing that:

   ```
   #include <windows.h>
   ...
   HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hStdOut, FOREGROUND_RED);
   std::cout << "Hello" << std::endl;
   ```

   For color constants list see: Console Screen Buffers - Windows Console
   See also: Colorizing text in the console with C++ - Stack Overflow

   Please note that alternative methods of coloring text suggested on the web, such as using color codes with system commands, may not always work. It is advisable to avoid these methods and, instead, use the approach mentioned above if you intend to add colors.

2. To get bonus points, you must indicate inside your readme.txt file the bonus additions that you implemented.

Self Decisions:
Any decision that you have to make, which does not have any clear guidance in the requirements, you can take a reasonable decision as long as it doesn't contradict any of the requirements or make the game naive or too restricted.

In case you have a question for which you don't have an answer and you don't want to make an assumption or you feel the assumption might be wrong or contradicts other requirements, please ask in the Exercise Forum in Mama.

Important – Important!!!
We will explain to you in one of the lab sessions how to use gotoxy for printing at a specific location on screen (X, Y), _kbhit and _getch for getting input from the user without blocking, and Sleep for controlling the pace of the game.
BUT, you don't have to wait for that, you can already watch now Keren Kalif explaining that in this great video tutorial: תכנות מונחה עצמים | קרן כליף | משחק הנחשים C++
The proper code would be published.

Exercise 2

In this exercise you will implement the following additions to your game:

Two tanks per player!

The game may have (depending on screen information, as explained below) multiple tanks per user. Switching control between tanks (moving control to the next tank) would not stop the tank's previous movement! Switching would be done using the following keys:

- Z (or z) for player 1
- M (or m) for player 2

Loading Screens from files

The game would look for files in the working directory, with the names *tanks-game*.screen* these files would be loaded in lexicographical order (i.e. *tanks-game_a.screen* before *tanks-game_b.screen* or *tanks-game_01.screen* before *tanks-game_02.screen* etc.). The files are text files (NOT binary files).

You MUST submit 3 screens with your exercise!

If there are no files, a proper message would be presented when trying to start a new game. The menu should have a new option to allow running a specific screen, by name.
The screen file should be a text file representing the screen, with the characters:

#	A "wall"
1	A tank that is owned by player 1, there can be multiple tanks for each player
2	A tank that is owned by player 2, there can be multiple tanks for each player
@	Mine
L	Indicating the legend top-left position, where Score and other relevant information shall be presented. It is the responsibility of the screen designer (not of your program) to make sure that the L is placed in a position not accessible by the game objects. You may assume that the screen designer follows this instruction. The size of the actual printed legend shall be not more than 3 lines height * 20 characters.

Note that the above chars are mandatory, even if you use other chars for drawing the board on screen (which is valid).

Position of the tanks' cannons when game starts would be random, but if the random position is illegal a new position would be drawn till there is a valid position. In case all positions are invalid (only if the tank is fully surrounded) then it will be born without a cannon.

Number of tanks for each player can be different.

A screen ends when one of the players loses all its tanks. Then the game continues to the next screen.

Score

Add score to your game. You should decide about the score formula.

Note

You should change your original code, to use new materials that we learned - where appropriate.

Exercise 3

Part 1 - Computer Player Mode
You should add an option for each player to be played by the computer.
Add the proper option in the menu, to allow the following options:

- Two human players: Human vs. Human
- Human vs. Computer (you can decide that in this option, the Human is always player 1, or you can just ask for each player whether it's human or computer).
- Two computer players: Computer vs. Computer.

Note: the minimal requirement for a computer player is to:

1. Try not to shoot its own tanks.
2. Try to shoot opponents' tanks when relevant.
3. Try moving a tank when a shell is chasing it (you can decide that this is the only trigger for moving a tank).

If you implemented a sophisticated computer player algorithm you can brag about it in the Bonus file, explaining your algorithm and asking for a bonus.

Note: the computer algorithm can get full read access of all game objects positions on the screen.

Part 2 - Saving and Reading Game Files
You should also implement an option to run a game from files and to record a game into files, mainly for testing. This would work as following:

- There would be a text file with a list of steps, per screen in a structure of your choice. The structure shall be concise, i.e. shall include only information that cannot be deduced from previous file info and the game rules. Do not add to the file information that can be deduced from previous data in file, for example there isn't a need to save into the file the tanks positions, only the change of directions. It is important to save each "step" with a "time" so it can be reproduced exactly as it happened. Since the position of the cannons on each screen start is not deterministic, you can save to the file a random generator seed, which would be created for each screen, using the same seed can guarantee the same random numbers are used when a screen is played from file. Note that each steps file is per a single screen, but for the two players and all their tanks.
- Name of file shall be in the format corresponding the screen name: *tanks-game*.steps* (e.g. *tanks-game_01.steps* for *tanks-game_01.screen*)
- There would be a file with the expected result for the screens, in the format corresponding to the screen name: *tanks-game*.result* (e.g. *tanks-game_01.result*), the file shall include the following information:
    - points of time* for this screen where a shell hits something on screen, what was hit and where
    - point of time* for this screen where a tank is dead by stepping on a mine
    - score gained for this screen by each player

* point of time = game's time counter, not actual time, each iteration can be counted as "1 time point"

You have the freedom to decide on the exact format of the files but without changing the above info or adding unnecessary additional info.

You should provide at least 3 examples, with the steps and results files (thus 9 files overall: screen, steps, results * 3 screens).

You should add a newly dedicated readme file called **files_format.txt** explaining your *steps* and *results* file formats, to allow creation of new files or update of existing files without recording.

The game shall be able to load and save your files!
All files shall be retrieved / saved from /to the current working directory.

Running the option for loading or saving game files would be done from the command line with the following parameters:

```
tanks.exe -load|-save [-silent]
```

The *-load/-save* option is for deciding whether the run is for saving files while playing (based on user input) or for loading files and playing the game from the files (ignoring any user input).
In the -load mode there is NO menu, just run the loaded game as is, **and finish!** Also you should ignore any user input, including ESC - there is no support for ESC in load mode!
In the -save mode there is a menu and the game is the same as in Ex2, except that files are saved. Note that each new game overrides the files of the previous game, i.e. we always keep the last game saved (you can always take the files and copy them to another folder manually if you wish). Saving a game should work for any combination of Human/Computer players. The file will be the same for human and computer player and when loading it there wouldn't be any difference (there isn't a requirement to present on screen whether the player is a human or computer, but if you want to add it then you can save this information to file).

The *-silent* option is relevant only for load, if it is provided in save mode you should ignore it, if it is provided in load mode, then the game shall run without printing to screen and just testing that the actual result corresponds to the expected result, with a proper output to screen (test passed or failed, and if failed - what failed). In silent mode you should avoid any unnecessary sleep, the game should run as fast as possible. Without -silent, the loaded game would be presented to screen, with some smaller sleep values than in a regular game.

Note: you should still support running your game in "simple" mode, without any command line parameters, as in Ex2: `tanks.exe`
In which case it will **not save or load files** and would behave as in Ex2 (except for the other addition of Ex3 below).

Exercise 4
Exercise 4 is a separate exercise built as a rehearsal for the exam.
It would be published separately.