# GAME311 Lab 1: Local Multiplayer Game C++ (week 1)

#### The purpose of this lab is to create a local turn based multiplayer game which will be used as a base moving forward to implement networking.

#### You are to create a 3x3 grid based game which contains some sort of pieces and is turn based between 2 players.

#### Players alternate turns.

#### The board should contain at least one piece for each player.

#### On the players turn they can choose to move the piece AND/OR attack with it.

#### Pieces should not be allowed to move or attack off the board

#### Attacks use a random number between 0 and 100 to calculate damage.

#### Pieces start with 100 health and when they go below 0 they are marked as dead.

#### When all of one players’ pieces are dead they have lost and the opponent wins.

#### Score should be tracked and players should be able to have unlimited rematches.

#### An example executable has been added to blackboard.

#### \*If you have an alternative idea for a game, you can pitch it to the instructor for approval.

#### Example Screenshots:

#### 

#### 

#### Getting Started ( week1)

#### Create a new solution with VS2017.

#### Follow these steps:

#### Open Visual Studio 2017

#### File 🡪 New 🡪 Project 🡪 Visual C++ 🡪 Empty Project

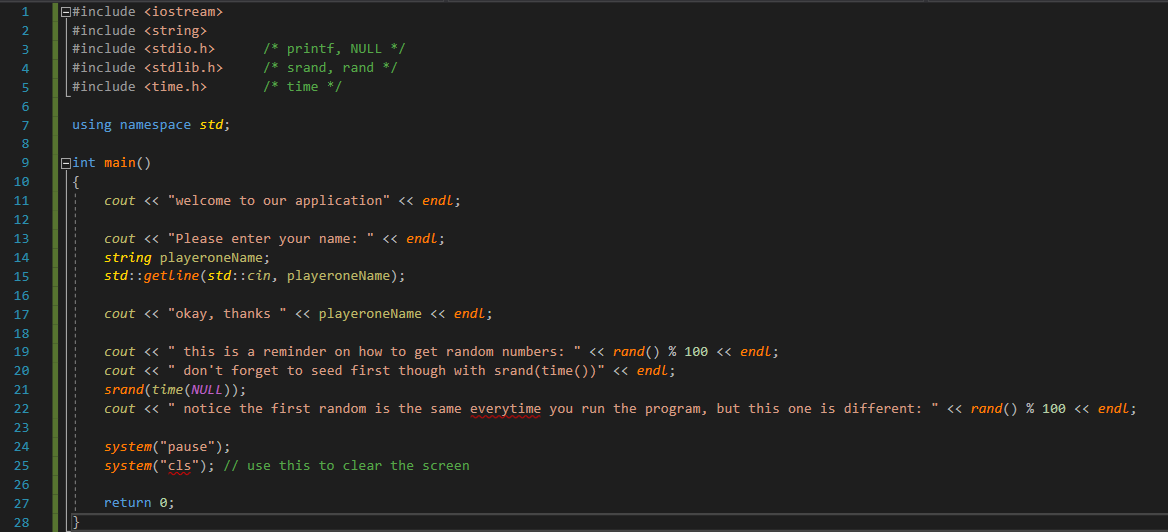
#### Make sure the Project name is Lab1-NetworkGame\_<BOTHNAMESHERE>,

1. Add a new source file to your project by right clicking on Source Files in the solution explorer on the right and selecting Add > New Item…
2. Name the File main.cpp

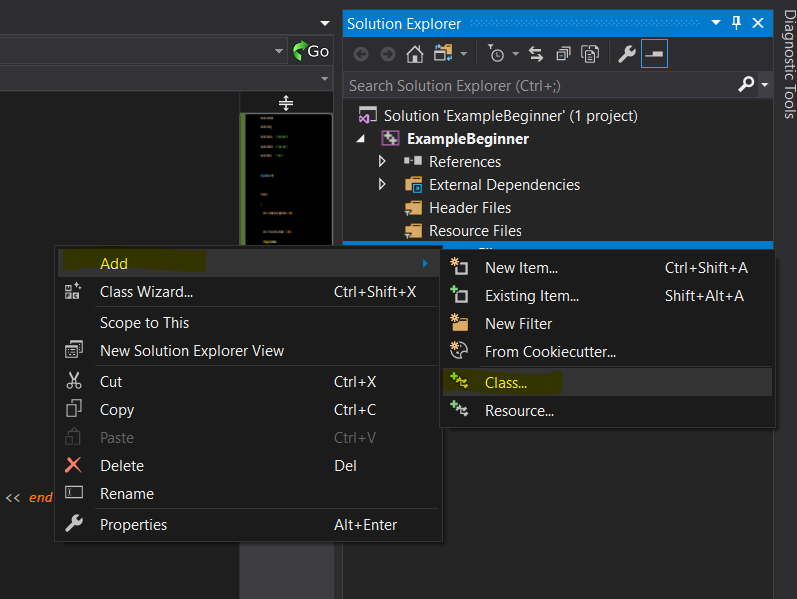
#### You are required to have a Program Header for each file with the name of the file and names of the programmers. Comment in-line for clarity. Example:

#### Add your basic C++ applications you used from first semester to create a base application.

As an example here is a refresher on a few things:



Add 3 classes to your project, both header and cpp file



**Game**

**Player**

**Piece**

Populate each of the classes with stubbed out functions to match the following UML diagrams:

**Game**

Behaviours:

[+] Constructor

[+] Destructor

[+] Overloaded Constructor with each player passed in as parameters

[+] StartGame function which resets both players game specific data by calling their individual StartGame() functions

// main loop for individual games this should be turn based going back and forth between the two players waiting for their turn to end

[+] CheckForWinner // checks for end of game

[+] ProcessTurn() // this should allow the user who is active to do any decision making logic required for your game.

[+] display board() // prints a visual representation of the board to players , this should always be displayed first on screen.

[+] display playerStats // prints out player specific information needed to make decisions

Properties:

[-] 2d Array of Player size 2 // keeps track of both players(passed in from the main game loop for multiple games)

[-] activePlayer; //current turn(which player)

**Player**

Behaviours:

[+] Constructor

[+] Destructor

[+] Overloaded Constructor with name passed in

[+] StartGame function which resets all the players information for piece(s)

[+] StartTurn resets any turn specific data or information needed.

[+] GetWins gets the number of times that player has won

[+] AddWins when one match is complete it increments the wins for that single player by 1

Optionals:

[+] EndTurn increments a turn taken counter

[+] IsTurnActive checks to ensure they haven’t completed actions for a turn

[+] GetActivePiece returns which piece is currently being moved or whatever other actions they can take

Properties:

[-] Piece\*[2]; // array of pieces if you have more than one

[-] name // players name

[-] wins // keeps track of wins

[-] gamesplayed // keeps track of number of games played

Optionals:

[-] moved // keep track of whether they have moved (OPTIONAL IF IT MEETS YOUR GAMES NEEDS)

[-] attacked // keeps track of whether they have attacked (OPTIONAL IF IT MEETS YOUR GAMES NEEDS)

[-] activePiece // keeps track of the active piece if you have multiple pieces

**Piece**

Behaviours:

[+] Constructor

[+] Destructor

[+] Overloaded Constructor with x and y location

[+] move the piece on the board

[+] takeDamage takes in a value for how much damage to take in

[+] getHealth getter for the health below

[+] getLocation getter for a vec2 location

[+] isAlive getter to check if the piece is dead or alive

Optionals:

[+] getLetter getter for the letter representation of the piece

Properties:

[-] health // health remaining of the piece

[-] alive // tells whether the piece is alive or dead

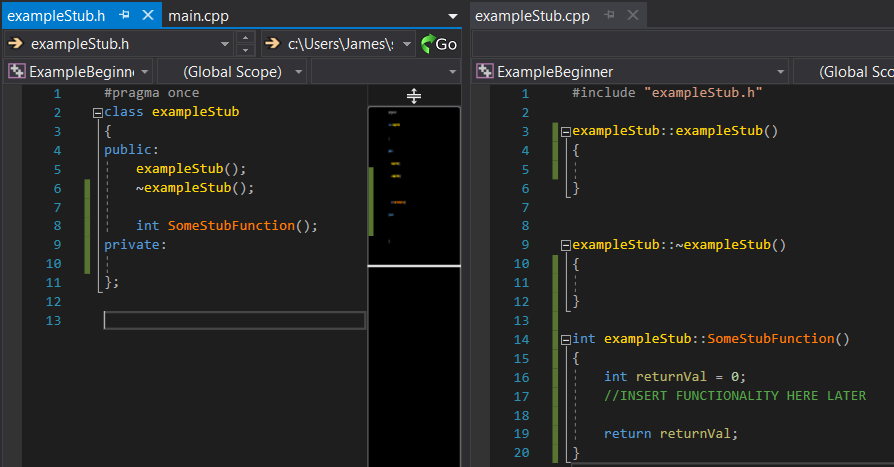
[-] location // x and y positions

Optionals:

[-] letter // letter to represent the piece.

Create both a cpp and header file stubbed out with each of the above functions.

Example of a stubbed function:



**Week 2 – Implementation**

The following is a breakdown of notes on how I started my implementation of my game. This is not a required flow for you to take, but it may be useful as a kickstarter to have steps to follow on how to start tackling the games code.

Main.cpp

* Implement the actual functionality of the functions stubbed out from week 1.
* Start by working your way in from the int main function in main.cpp.
* Create an int main() function that returns 0; in the main.cpp file
* intialize random using srand() inside your int main
* print out some sort of welcome message using cout
* prompt each of the users to provide a name
* take in two names from the players using getline() [ include #include <stdio.h>
* supply in the names of the players to the constructor of 2 different player objects made from the Player class
* this should automatically create the any pieces required for the game within it's own constructor.
* Create a main loop here ( this will be a loop to handle multiple matches)
  + Create an instance of a game class and pass references to both players into the constructor
  + call startGame() on the game object
* this should loop until one player wins, then it should display the results and print out both players names as well as how many wins each of them have.

Game.cpp

- StartGame() function, create a match loop.

- call startgame on both players instances to reset all their pieces.

- Inside the match loop it should randomly assign which player is the active player using rand and limiting it to 0 or 1.

- use cout to indicate which player is active.

- print the board out so the players can see where the pieces are. (call displayBoard())

- after assigning the active player, process that players turn (call ProcessTurn()).

- increment the activeplayer value.

- if it's over 1 set it back to 0.

- have it call the checkforwinner function to see if the game is over.

- this function will essentially check the health of both of each players pieces to ensure they aren't 0 or below.

- if they are, they lose and the match is over.

-end loop

ProcessTurn()

- prompt the user to make a choice for their actions which should be written out to them

- perform a while loop which loops until they have completed the actions required for a turn.

- use a prompt with cout and a retrieval with cin.

- use an enum for ACTIONS the user has such as for MOVE, ATTACK, END\_TURN.

- create a switch which evaluates the users choice.

Depending on your specific gameplay these may be completely different.

* + - * + You may need multiple functions to handle this segment of the users options
      * Examples:

- if the user chose MOVE, present them with their options for movement.

- up down left right (another switch and enum)

- if they chose attack process that…

Displayboard() – Create a 2d array of chars and initialize it entirely to '-'

- Populate the 2d array using both players pieces locations and set them to Upper case A and B for player one's pieces.

- I used X & Y for player two's pieces.

- Use 2 nested for loops to iterate through the array and print out the contents.

- use \t to space them out evenly with tabs

**Week 3 - Finalizing**

This week should be spent finalizing the game and fixing any minor bugs remaining. The game should be playable at the end of this week with the ability to run multiple matches of a turn based game which tracks wins for each user.

**MARKING:**

Gameplay : **50%**   
 - Match loop – 7.5%  
 - Game Loop – 7.5%   
 - Turn Loop – 10%  
 - actual gameplay implementations – 25%  
 -mechanincs 10%  
 - functionality 15%  
Required Classes: **30%**  
 - meets required functionality – 15%  
 - used appropriately – 15%  
Architecture of Code: **15%**  
 - implementations, optimizations, function use, class design, efficiency, coding standards, comments  
symmetry / cleanliness / aesthetics **5%**

**DUE DATE:**   
Tuesday January 29th 2019.