Battleships

Must have:

- Points on a grid (10 x 10)
- Two grids for each player
  - One which shows their own ships and status
  - One which shows empty enemy grid with status from shots fired (hit or miss)
- Ability to place ships on grid from a starting point to ending point
  - Must be either horizontal or vertical
  - Cannot overlap ships
- Store ship locations
- Ability to fire at locations of other players grid
- Feedback from each shot fired
  - Miss
  - Hit
- Notify a player when they have destroyed an enemy's ship
- Players take turns at firing shots

Should have:

- Able to handling errors from user inputs
- Game ends when all player's ships have been destroyed
- Game instructions

Nice to have:

- A main menu
- Player statistics
  - Hit/miss ratio
  - Win/loss ratio
  - Games played
- Enemy AI to take computer turns
- Games saved – to be continued later

Idea of what it could look like?

```
Your grid with your ships:

[    ] | [ A ] [ B ] [ C ] [ D ] [ E ] [ F ] [ G ] [ H ] [ I ] [ J ]
-------------------------------------------------------------------
[  1 ] | [ X ] [   ] [ X ] [ X ] [ X ] [ X ] [   ] [   ] [ X ] [   ]
[  2 ] | [ X ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [ X ] [   ]
[  3 ] | [ X ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  4 ] | [ X ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  5 ] | [ X ] [   ] [ X ] [ X ] [ X ] [   ] [   ] [   ] [   ] [   ]
[  6 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  7 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  8 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  9 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  10 ] | [   ] [   ] [   ] [   ] [ X ] [ X ] [ X ] [   ] [   ] [   ]

Enemy grid:

[    ] | [ A ] [ B ] [ C ] [ D ] [ E ] [ F ] [ G ] [ H ] [ I ] [ J ]
-------------------------------------------------------------------
[  1 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  2 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  3 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  4 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  5 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  6 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  7 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  8 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  9 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]
[  10 ] | [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ]

Its your turn. At which column do you wish to fire at? A

Which row? 1
```

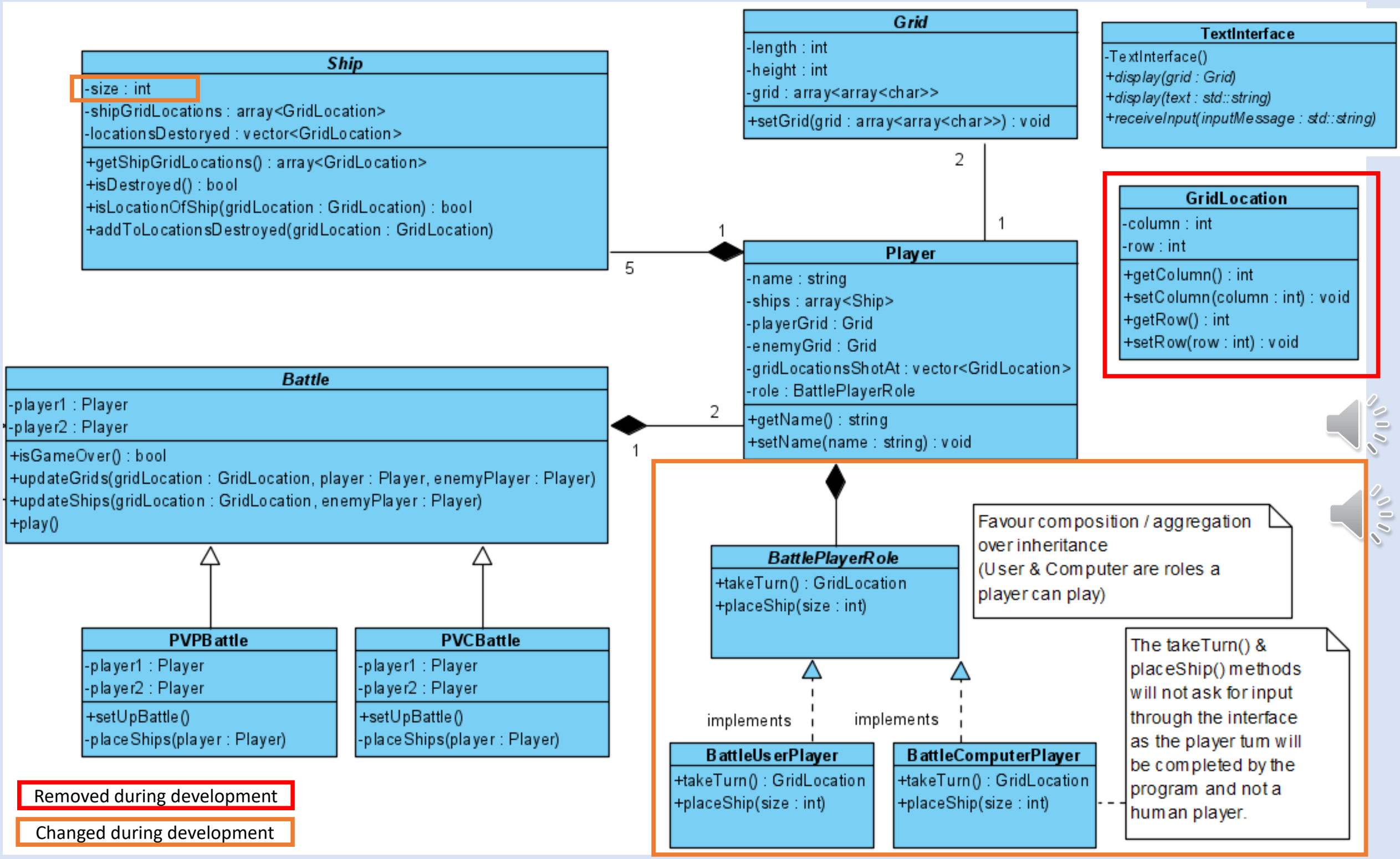
Variable / Function / Class Ideas (Starting Ideas)

- player1 & player2 – Player object to encapsulate player's grids and actions
  - ownGrid: vector< vector < char > >
  - fireShot(column : char, row : int)
- game – Game object
  - setUpGame (player1Grid, player2Grid)
  - placeShips (player : Player)
  - gameIsFinished ( ) : bool

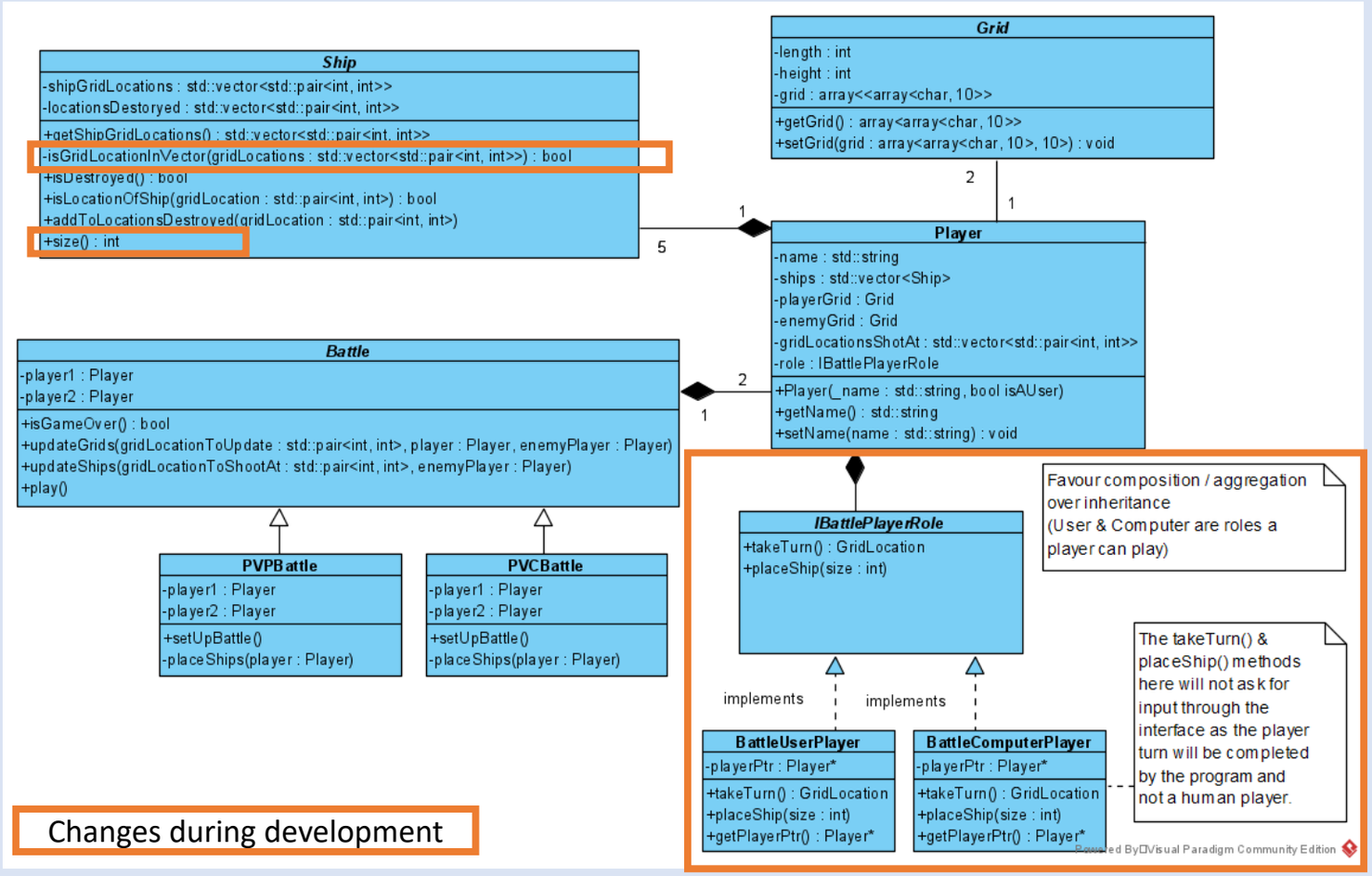
Architecture Design

Planning with nice-to-have features in-mind:
The Player vs Computer mode is not a must-have or should-have feature, but I have considered how this could be implemented as I wanted a structure that would allow this functionality to be added with minor changes to the rest of the code.

Initial architecture design before starting development (coding)



Architecture Design – Changes



Changes during development

<u>Minor changes during development</u>

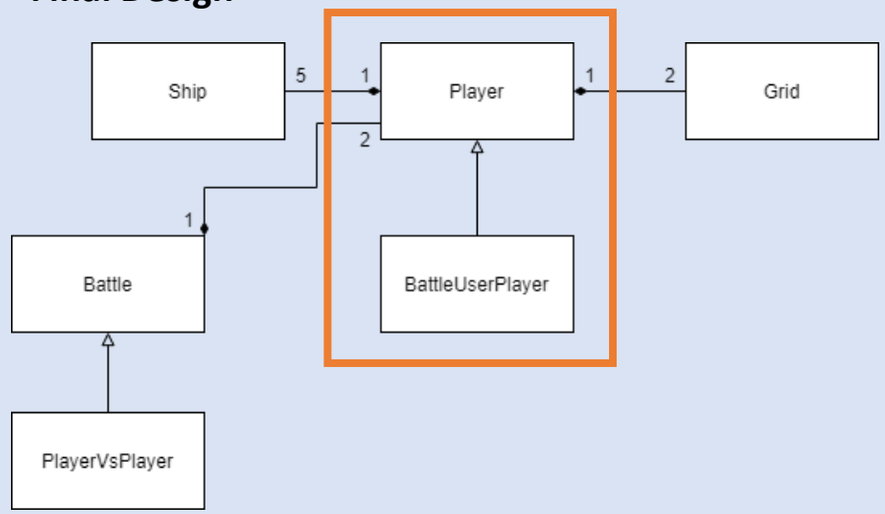
- During development I had to make a lot of small changes to improve the system.

- The **GridLocation** class was **unnecessary.** I decided to use the **std::pair** data structure to pass 2 integers around my system instead.

- size variable changed to a method
  - Calculated from number of locations belonging to a ship
- I will remember to consider this in **future projects**
  - No need to store data which can be calculated

- I also realized that I needed to store a **pointer** to the player object in the classes which implement the BattlePlayerRole 'interface' class. Which are discuss on the right hand side of this slide.

Final Design



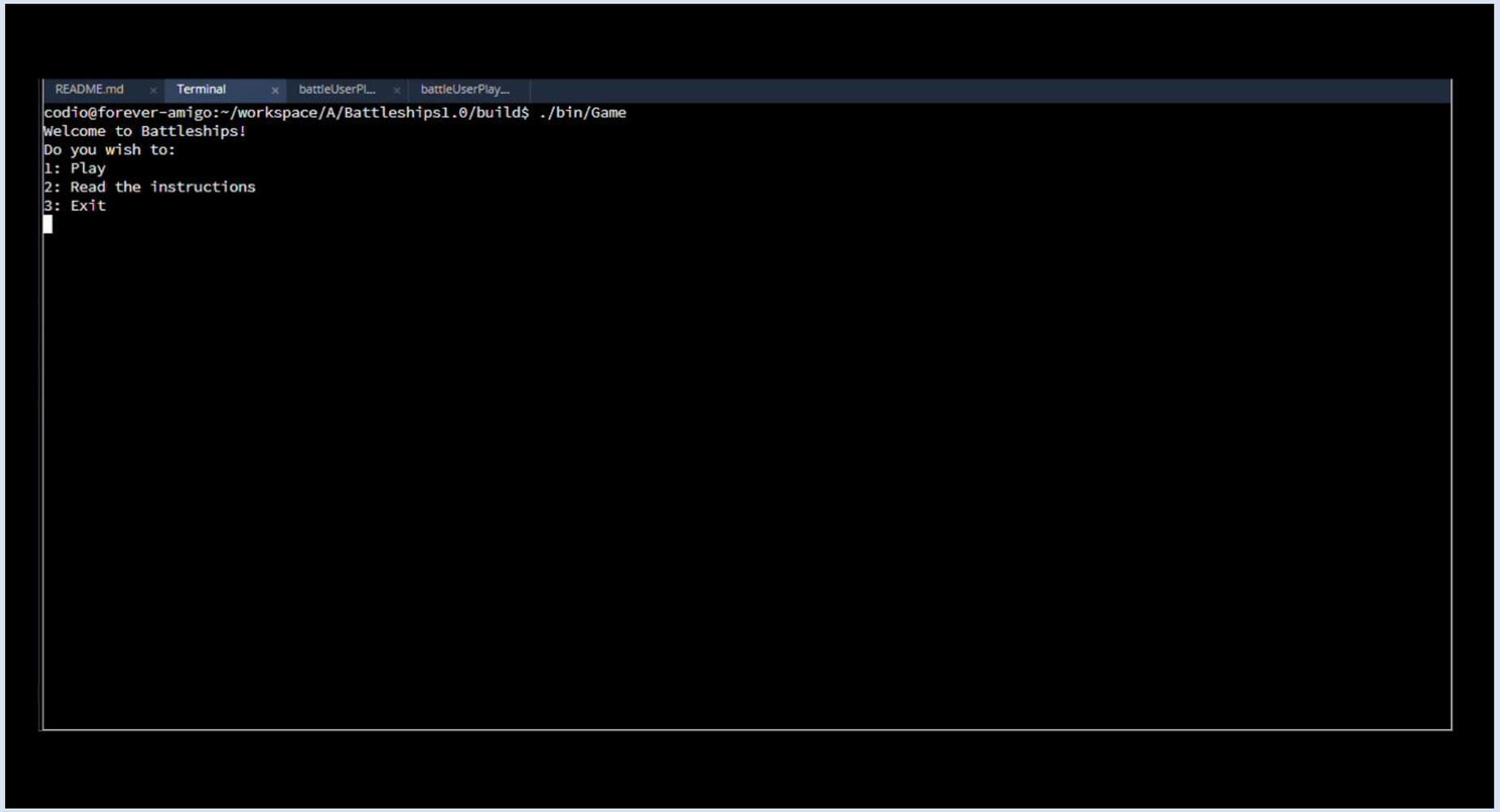
I reaslised that realistically a human could not switch to the role of a computer player so this 'role' for the player unnecessarily overcomplicating my game design. With the takeTurn() function (the function which will get the location a Player wishes to fire at on the enemy's grid) I had to access members from the Player object. This meant that the 'role' object member in the Player object needed to store a pointer back to the Player object in which it is stored to acess the Player object's members.

I was having a lot of segmentation faults (core dumps) when trying to implement my initial design and my time was running out for the project. So I evaluated the problem and decided to alter this part of my architectural structure to consist of inheritance instead.
For the situation, changing my design was the most effective solution as it meant I was able to implement the functionality and move on to the rest of the game.

This choice enabled me to be able to move on and complete all the must-have and should-have features of the game in time for the project deadline.

MVP Demonstration

Core functionality demonstrated. The game allows both players to place 5 ships at the start. After this, each player will take turns to fire at each other's grids until someone has destroyed all of their enemie's ships.



MORE CHANGES MADE DURING DEVELOPMENT:

- I changed the **rows and columns** to be numbered **0 to 9 instead of A-J & 1-10**. The first index of an array is 0 so when I get the user to enter rows and columns from 0 to 9 I can directly use the integers they have inputted to access the data.
- Storing **smart pointers** to objects which are within other object instances instead of storing copies
  - E.g. A Player object has 2 Grid instances which I now am storing as pointers
    - Enemy Grid
    - Their Own Grid
- **Fixed infinite loop** caused by the user entering in anything other than an integer when asked for an integer
- Added **Menu** to allow user to access **instructions**. I have not written the instructions but have provided the functionality necessary for a user to access them.

Must have:

- Points on a grid (10 x 10) ✔
- Two grids for each player ✔
  - One which shows their own ships and status ✔
  - One which shows empty enemy grid with status from shots fired (hit or miss) ✔
- Ability to place ships on grid from a starting point to ending point ✔
  - Must be either horizontal or vertical ✔
  - Cannot overlap ships ✔
- Store ship locations ✔
- Ability to fire at locations of other players grid ✔
- Feedback from each shot fired ✔
  - Miss ✔
  - Hit ✔
- Notify a player when they have sunk an enemy's ship ✔
- Players take turns at firing shots ✔

Should have:

- Able to handling errors from user inputs ✔
- Game ends when all player's ships have been destroyed ✔
- Game instructions ✔

Nice to have:

- A main menu ✔
- Player statistics
- Enemy AI to take computer turns
- Games saved – to be continued later

Explanation of Code

This might not be my most complex piece of code I wrote but I wouldn't have enough time to explain them.

This method deals with updating the Grid and Ship instances accordingly after a location has been chosen for a player's turn.

The boolean **return value** is used outside of this method to notify the Player if they have **"HIT"** or **"MISSED"** on their turn. This is found in the takeTurnForAGivenBattleUserPlayer method inside the same class.

```
if(updateGridsAndShips(shotTakenP1, player, enemyPlayer))
    TextInterface::display("HIT");
else
    TextInterface::display("MISS");
```

```
/* Method to update the Grids and Ships after a location has been choosen for a player's turn */
/* Returns - true if ship was hit, false if a ship was not hit */
bool PlayerVsPlayer::updateGridsAndShips(const std::pair<int,int> &gridLocationToUpdate, const Player& player, const Player& enemyPlayer)
{
    // Updating the Player's status grid for the shots they have fired
    bool shipWasHit = false;
    bool shipWasDestroyed = false;
    for(std::shared_ptr<Ship> enemyShip : enemyPlayer.getShips()) // For each Ship belonging to the enemy player
    {
        if(enemyShip->isLocationOfShip(gridLocationToUpdate) && !enemyShip->isLocationAlreadyDestroyed(gridLocationToUpdate)) // If the player hit a ship
        {
            enemyShip->addToLocationsDestroyed(gridLocationToUpdate); // Add to locations destroyed on the ship instance
            shipWasHit = true;
            if(enemyShip->isDestroyed())
                TextInterface::display("Ship destroyed!");
        }
    }

    int row = std::get<1>(gridLocationToUpdate);
    int col = std::get<0>(gridLocationToUpdate);

    if(shipWasHit)
    {
        player.getEnemyGrid()->alterGridPositionChar('H',row,col); // Players status grid of enemy's grid
        enemyPlayer.getOwnGrid()->alterGridPositionChar('H',row,col); // Enemy's own grid
    }

    else
    {
        player.getEnemyGrid()->alterGridPositionChar('M',row,col); // Players status grid of enemy's grid
        enemyPlayer.getOwnGrid()->alterGridPositionChar('M',row,col); // Enemy's own grid
    }

    return shipWasHit;
}
```

Challenges & Reflection

All of the changes discussed on slide 3 to my initial architecture design were the biggest challenges I faced. Applying my knowledge gained from 4001CEM (Software Design) and independent reading helped me to produce these detailed diagrams.

- Working on this project on my own was difficult yet rewarding
  - My project was limited by my knowledge and my understanding
  - I was able to apply skills necessary to complete the project that were developed through my own independent study

- Using **suitable** variable and functions **names**
  - Making code easier to read and understand
  - I want to continue to use appropriate names to ease understanding of my code in **future projects** as this is very important whilst working as part of a team

- Using **Codio** as my **IDE**
  - Needed to independently learn linux terminal commands
  - Will be able to use Codio and **Linux** for future development projects

- Applying and using **CMake** build system
  - Learnt the basics from 4003CEM lab tasks
  - Adapted the code for my own project
  - I will use CMake as a build system for future projects as it simplifies the compilation of multiple files

- Completed all must-have and should-have features
  - My thorough initial **planning** and **design** was key to implementing these
  - I will ensure planning and design are completed to high standard in future projects
    - Projects will be completed on time with requirements met if appropriate planning is completed
  - Also allowing **change** to **requirements** was a key factor in completing these

- The use of **Visual Paradigm** was beneficial for system archetectural design
- I will continue to use this software to produce quality diagrams