MET cS665 Summer 2 2018

Abel Gancsos

Abel Gancsos productions | Hawthorne, NJ

AMGBattleship v. 2

2018

Contents

1 Application Description 3

1.1 Implementation Details 3

1.2 Assumptions 3

1.3 Prerequisites 3

2 Assignment Tasks 4

2.1 Refactoring 4

2.2 Implementation Description 5

2.3 UML Class Diagram 6

2.4 UML Sequence Diagram 7

Bibliography 8

Appendices 9

Appendix A – Setup 9

Appendix A – Building project 9

Appendix B – Running executable 10

# 1 Application Description

## 1.1 Implementation Details

*The following details are also available in the README.md file*

This command-line utility serves as a text-based console version of the classic Battleship board game. Typically there would be two users that would have the same number of ships on the board at different locations and they both take turns trying to sink the other users ships. In the real game, each ship would have a different length and thus a different number of required shots to sink it; however, in this version of the game, every ship is represented by the same size, but in different locations. After each shot, the current player will only see there board, locations of their ships, what shots were taken against their ships, and which ships were hit/sunk. The board is then cleared in preparation for the next player.

## 1.2 Assumptions

* Users play the game on the same command line, but their own context during their turn.
* The game may add other attacks in the future.
* There will only be 2 players at a time on the board.
* A ship will only take up one block on the board. This may change in future releases.
* The ship count can not and should not be larger than the map size.
* Both players will start with the same number of ships.
* The board width and height are the same for each player.
* The values for the board height and board width are for each side/player (not the total board).
* The current player will only see their own board.

## 1.3 Prerequisites

* C++ compiler with C++11 capability
* CMake version 3.11 or higher
* XCode command-line tools (MacOS only)
* Visual Studio 2017 or higher (Windows only)

# 2 Assignment Tasks

## 2.1 Refactoring

As was mentioned for the first release of AMGBattleship, certain items were intentionally left in the initial release that may have been considered messy or not best practice. For example, one item that was declared to be an issue in the initial release was that the implementation not only really used one client (command-line session) to manage the game board, which required that both players be located in the same physical location, but the whole game was encapsulated in the main system. This combination then required multiple attributes in the client to handle both players, which made the transferring of the data between components unsafe and messy. Ideally, the main system would be in a completely separate location and the board would become the true client, which would be split up into multiple instances. Then using an AMGBattleshipData class to hold the games data, each client would build it’s own requests and therefore allow for more security as well as flexibility. These items were intentionally left in the first release because the product owner decided that shipping a functional product in a timely manner was more important than having perfect code. This release focused primarily on code cleanup as well as stabilizing the product.

This release focuses on addressing the issue where one board was used to manage both players data. In order to do this, the data also had to be made separate so that it could be easily transferred between components, thus the AMGBattleshipData class was added to the product. At this time, each board, which by current design is considered to be the true client, contains it’s own data object to maintain the players information. As the current release still uses one main system to manage both players, this then puts the system in a better position to implement a client-server game play because the data is separate from the main system and is apart of the client.

A third item that was focused on during the refactoring process was general code cleanup and stabilization. The way that this was approach was that any duplicated methods were removed and any undocumented code was corrected with documentation. Certain methods do seem to be duplicated; however, the information that is involved is under a different context. For example, the AMGBattleshipBoard. SetShipCount() method exists to set the number of games on the player’s board, which the AMGBattleshipData. SetShipCount() method is used to calculate the attacks as well as if the game play has completed. Another benefit of certain duplicated methods is to ensure that the data is safely set without a risk of injection.

With any refactoring, there is a risk for the system to change in behavior, especially if the changes are significant. As the data had to be split apart to allow multiple client instances was not a small change, despite it being implemented in a timely manner, the behavior of the system could have changed. During testing, no severe change to behavior has occurred, one change that was observed is that while generating the ships for the players, there is a one or two second delay. Otherwise, the data is displayed as it did in the initial release, which by design would display the player’s ships, correct markings if they were hit or not, and then the opponents shots. It was by design that the current player is unable to see the opponent’s board because by design of the original Battleship game only offers the location of where the player shot and if it was a hit or miss. It does not offer a visual representation of the opponent’s board and thus AMGBattleship by design will only list where the player shot on the opponent’s board.

## 2.2 Implementation Description

*The following details are also available in the README.md file*

Similarly to the first release of the AMGBattleship game, this release is strongly based on the Command design pattern because the end result very much behaves as an “attack” command from one player to another. By using the Command method, all the information can be packaged into one object, which the invoker ends up performing the operations, in this case shooting at a target on the opponent’s board. Currently, the game still uses one main system to prompt the player for the coordinates; however, starting in this release, there are at least two separate display adapters, AMGBattleshipBoard instances. Ideally, the display adapters would be completely independent clients and the AMGBattleship main system would simply be the server, ie client-server relationship. Note that like the initial release, this release will only print the current player’s board, but will provide a list of attempted shots from either player. At this time, it does not state if the shot was a miss, but should be displayed on the board with the appropriate board indicator. This is by design because in the original Battleship game, the opponents board is never displayed to the current player, unless it is the end of the game.

In terms of flexibility, the current release offers the same level of flexibility when it comes to commands. This is to say that the AMGBattleshipBroker invoker class supports different types of attacks, so if a new attack or command is to be added, it just has to implement the AMGBattleshipCommand abstract class and the target may also need to be updated with the new operation. Apart from the existing command flexibility, because the board for each player was split into separate objects, this allows a safer and cleaner way to output the game status to each user. The main reason and example of how this allows for more flexibility is because in a future release, the game would be refactored to use a client-serer mechanism, allowing players to interact with the game in remote locations. Having the displays as separate instances, it puts the game into a better position to implement the end architecture. Another example of the flexibility of the latest release is that the data itself was split into it’s own separate object, AMGBattleshipData. Currently, this is held within AMGBattleshipBoard instance as a way of shipping the data with the player; however, because the data is now a separate object, it could be moved into any context necessary.

When it comes to simplicity and understandability, there were attempts made to make the documentation clearer in order for future developers to understand what operations each component performs. Another example of the simplicity, particularly in this release of AMGBattleship, is that because the data was split into it’s own object, it is easier to modify when needed, but is also protected against inappropriate injection. The packaging of the source code was left exactly the same in order to avoid inconsistencies as well as any breaks in the architecture.

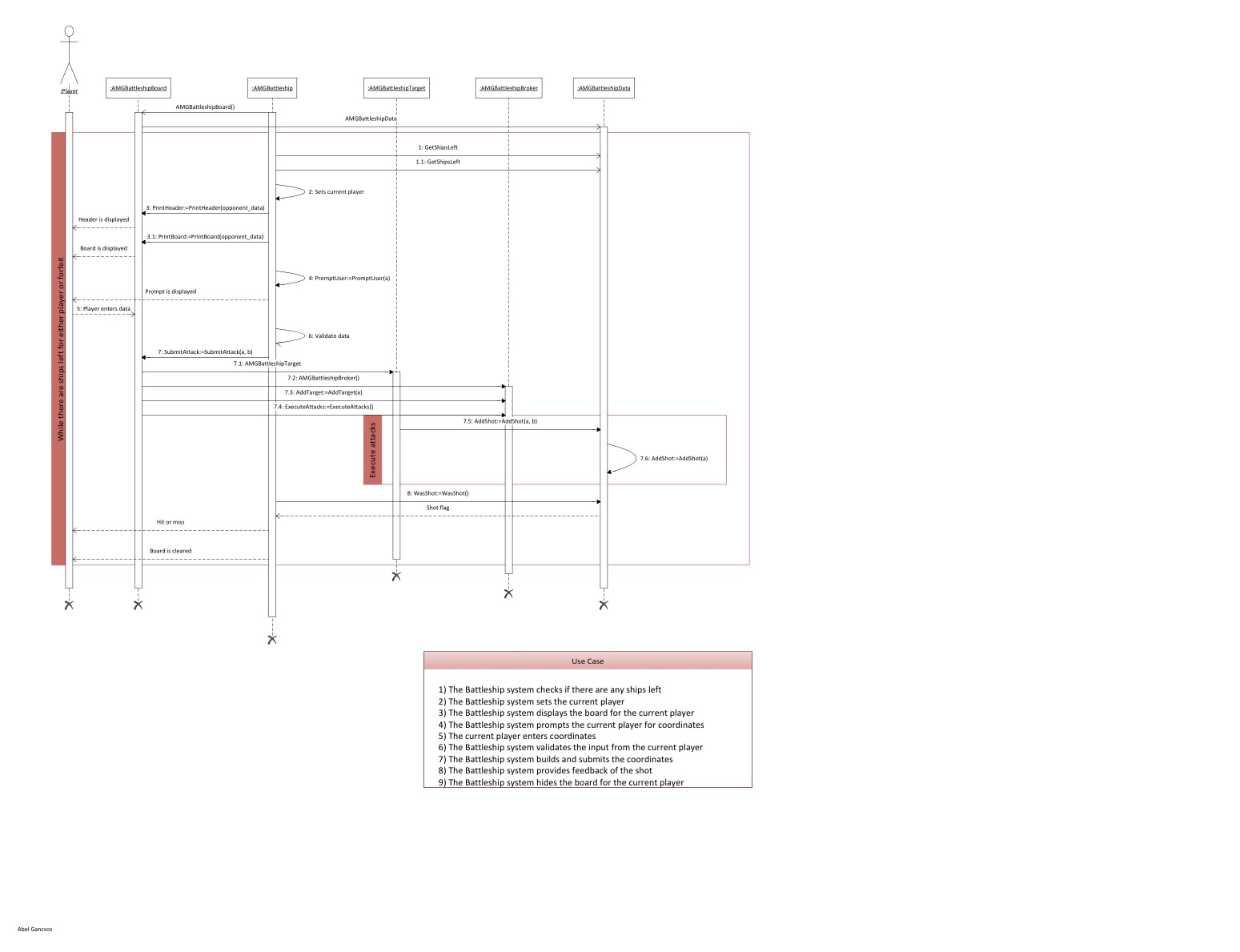
In regards to duplicate code, because the data was moved to a separate object, some unused or duplicated methods were removed. However, some methods were left in multiple locations in the source in order to protect that data and properly manage the environment of the game. One note with mentioning is that because the main AMGBattleship class is still managing the game play, some areas of the source, particularly in this class use switch-case statements based on the current player to know which player is performing the attack. Writing a separate method for this tasks may not have provided much benefits as the switch-case blocks are quite small and to build methods would add unnecessary code.

So in brief, this release focused on separating the displays for the players as well as the data in order to move towards a client-server architecture. Additional changes involved removing inappropriate or duplicated code and adding clearer documentation for the source. Although the first two changes seemed large, thankfully there were minimal behavioral changes to the final product.

## 2.3 UML Class Diagram



## 2.4 UML Sequence Diagram



# Bibliography

Döring, A. (2016, 11 07). *minimal\_cmake\_example*. Retrieved 07 03, 2018, from GitHub Web site: https://github.com/krux02/minimal\_cmake\_example

Gancsos, A. (2018, 07). AMGBattleship. Hawthorne, NJ, USA.

Google Corporation. (2018, 06 19). *Google C++ Style Guide*. Retrieved 07 03, 2018, from Google GitHub styles: https://google.github.io/styleguide/cppguide.html

Kitware Corporation. (n.d.). *Download*. Retrieved 07 03, 2018, from CMake Web site: https://cmake.org/download/

Lucid Software Corporation. (2017). *Class Diagram Tutorial*. Retrieved 7 25, 2017, from Lucid Software Corporation Web site: https://www.lucidchart.com/pages/uml/class-diagram

Microsoft Corporation. (2010). Visio. (2010). Redmond, WA, USA.

Oracle Corporation. (2017). VirtualBox. (5.1.18). Redwood City, CA, USA.

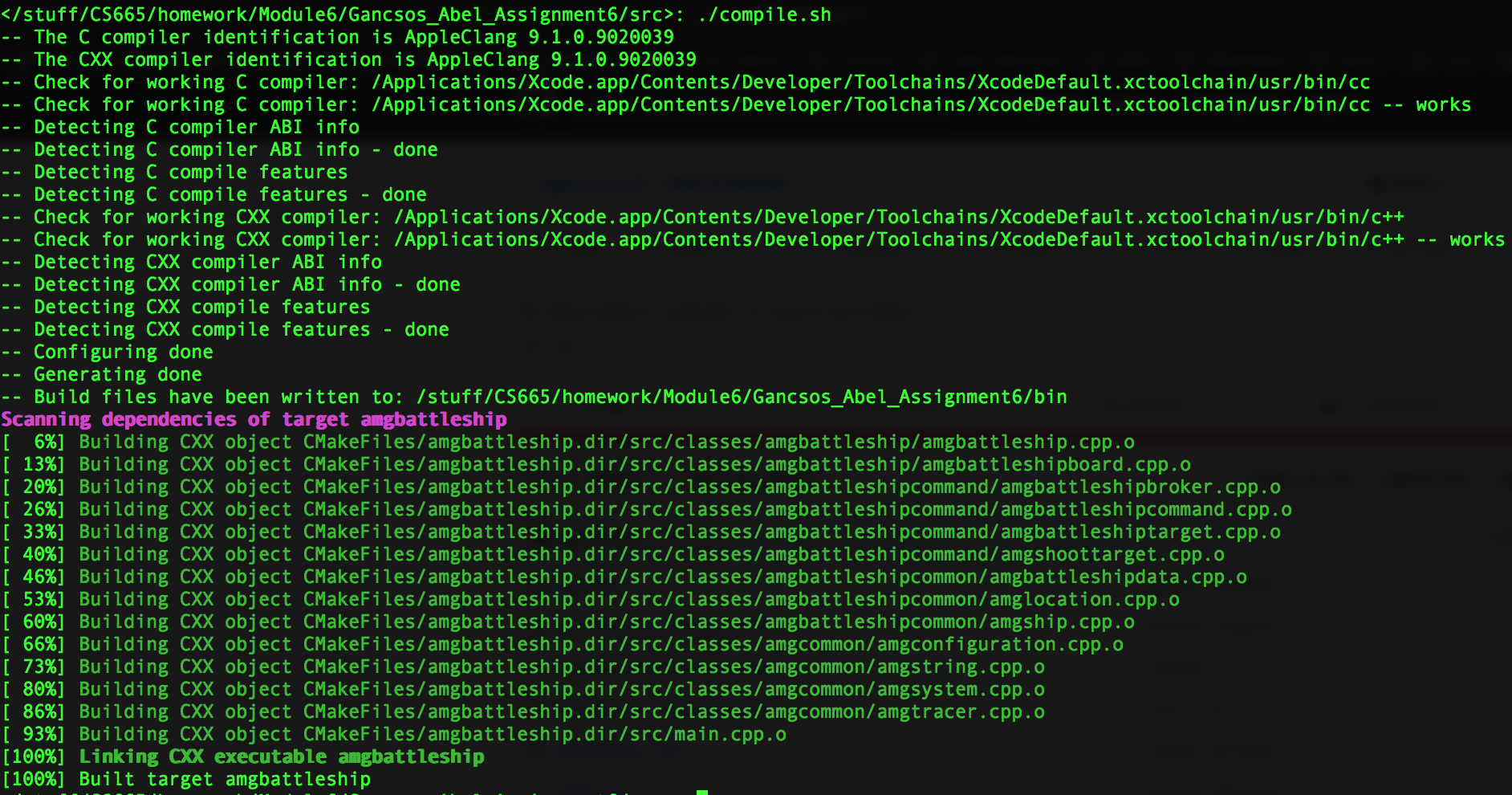
Tutorials Point. (2018, 07 14). *Design Patterns Command Pattern*. Retrieved 07 24, 2018, from Tutorials Point Web site: https://www.tutorialspoint.com/design\_pattern/command\_pattern.htm

# Appendices

## Appendix A – Setup

The project can be built using the run.sh or run.bat scripts, depending on the platform. Both of these scripts will run cmake to generate the CMakeFiles and then run the make command in the bin directory. Alternatively, the project can be built manually using the CMakeLists.txt file that comes with the package.

## Appendix A – Building project



## Appendix B – Running executable

