MET cS665 Summer 2 2018

Abel Gancsos

Abel Gancsos productions | Hawthorne, NJ

AMGBattleship

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 Implementation Description 4

2.2 UML Class Diagram 5

2.3 UML Sequence Diagram 6

Bibliography 7

Appendices 8

Appendix A – Setup 8

Appendix A – Building project 8

Appendix B – Running executable 9

# 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 Implementation Description

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

Due to it’s simplicity as well as it’s validity to the use in the program, the implementation primarily uses the Command design pattern. The thinking behind the decision to use this pattern is that a player might not know the coordinates for an opponent’s ships, so when they decide on the coordinates to shot at, the requested target would be sent as a command to the system. The system, which ideally would be a remote component, would invoke the command, which would extract the data that was packaged in the request and then calculate if a ship was hit.

In terms of flexibility, the current version of the implementation could be designed to be more scalable; however, there are indications that new types could be easily added. For example, because the implementation is based on multiple abstract classes, such as the AMGBattleshipCommand class, new attacks could easily be added if the game designer decided to add new weapons.

One example of how the design could be improved in a future release is that because the current implementation really uses only one client (command-line session) to manage the game board, both players would be required to be located in the same physical location. This also required multiple attributes in the client to handle both players. 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.

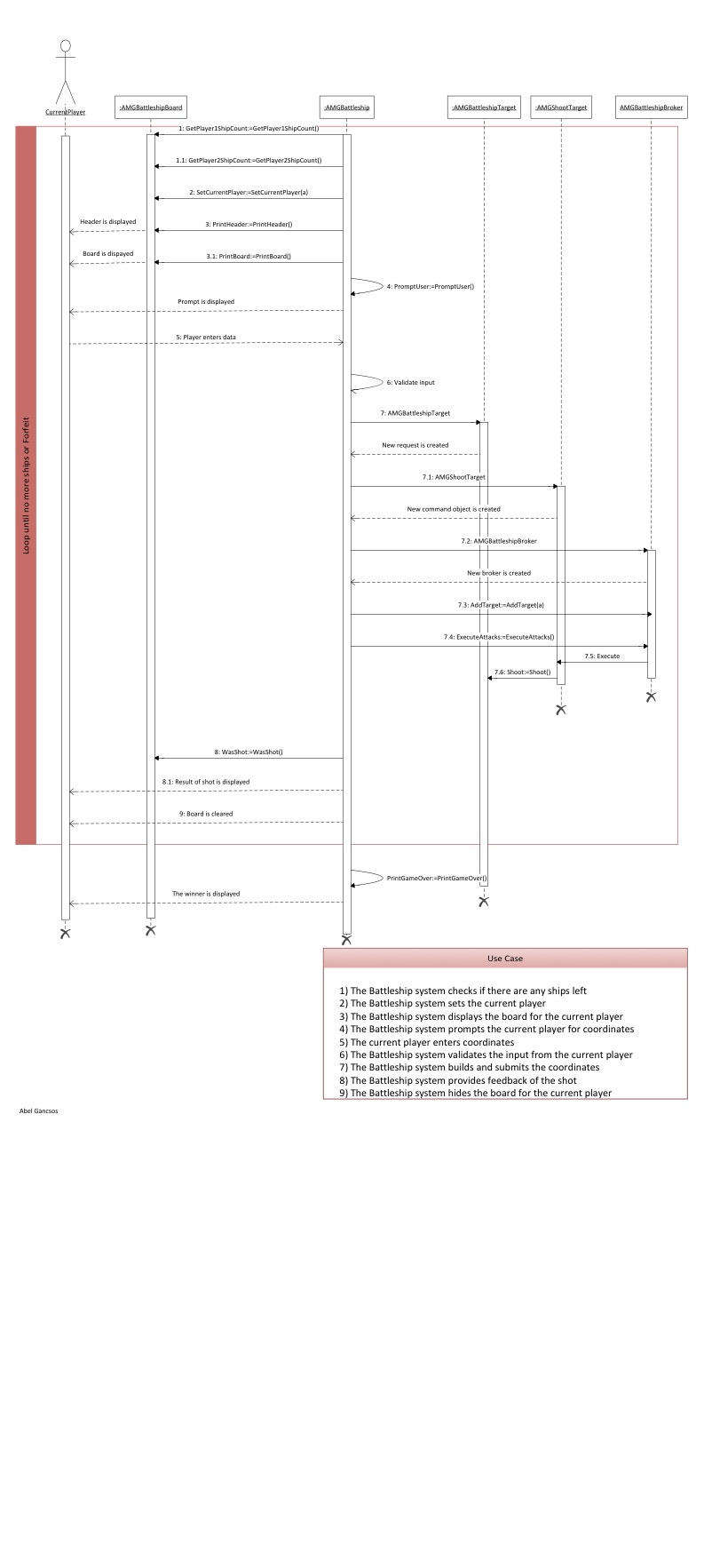
When it comes to how simple the system is to understand, particularly by other developers, it does have some complexity as the implementation is of a game; however, there are indicators of simplicity. For example, when taking a closer look at the implementation, one might notice that there’s really only two main packages. One of them is the amgbattleship package, which includes the main classes that drive the program as well as the board class. Then there’s the amgbattleshipcommand class that contains the base command class, AMGBattleshipCommand, which is used to build out the actual AMGShootTarget command that adds the new shot to the player’s shots collection as well as calculates if it was a hit or miss. In addition to the command classes, the command package also contains the AMGBattleshipTarget class, which acts as the request object that contains the data for the command. Then there’s the AMGBattleshipBroker class that actually invokes a collection of commands, in this case always has a single object. In the future, multiple attacks might be added to this collection in order to invoke combinations or “combos”. There’s also two common packages, of which the amgbattleshipcommon package contains basic types that help build out the request. Another note in terms of understandability is that the code was documented so that other developers would know which methods perform which operations.

In regards to avoiding duplicated code, for the most part, the implementation uses several methods, particularly in the AMGBattleshipBoard class, for code blocks that are used in multiple areas. However, some code has been duplicated, such as the SetBoardWidth(), SetBoardHeight(), and SetShipCount() methods, for security purposes. For example, the implementation has one copy of these methods for the AMGBattleship “session” or wrapper class so that the main method can safely set those fields from the command-line arguments. Then from those copies of the methods, the fields are set on the AMGBattleshipBoard instance to configure the board.

## 2.2 UML Class Diagram



## 2.3 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

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.

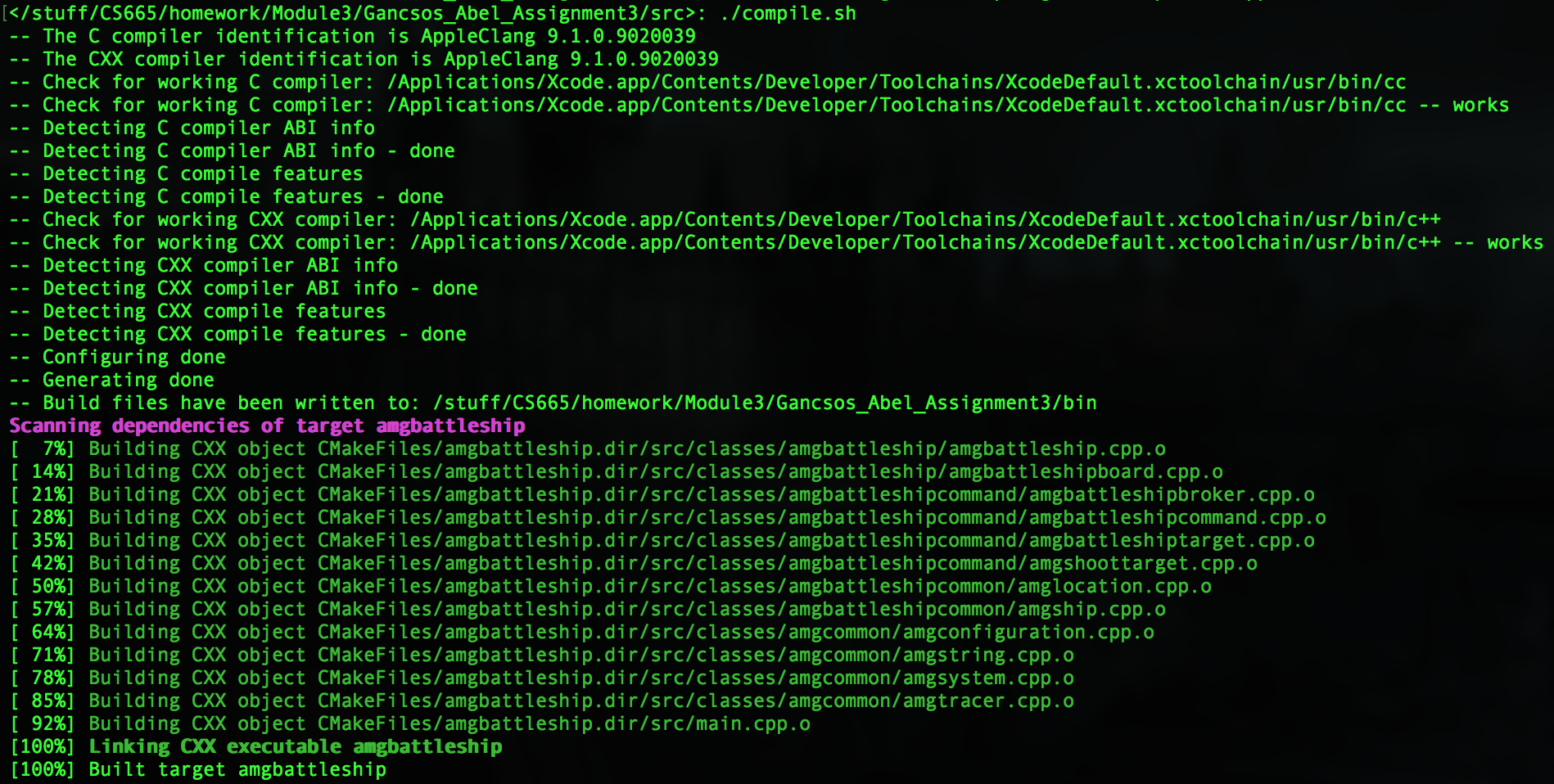
Tutorials Point. (2018, 07 14). *Design Patterns Command Pattern*. Retrieved 07 24, 2018, from Tutorials Point Web site: https://www.tutorialspoint.com/design\_pattern/command\_pattern.htm

# 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

