Submitted by:	
Marker:	
General Notes:	

Table:	Value	Max.	Value		
	0%	100.00%		1	**Part 1 - *CommandProcessor and command processor adapter***
	0%	10.00%		1.1	**Knowledge/Correctness of Game Rules**
	0%	0.00%	0	1.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	0%	10.00%		1.2	**Compliance of solution with Stated Problem**
	0%	0.00%	0	1.2.1	Upon starting the application, a command line option is set to either read commands from the console or from a given file.
	0%	0.00%	0	1.2.2	Commands are read from the console or from the given file.
	0%	0.00%	0	1.2.3	Commands can be read from a file using the **CommandProcessorAdapter** class that is a subclass of the **CommandProcessor** class.
	0%	0.00%	0	1.2.4	Commands are saw in a list of commands in the command processor.
	0%	0.00%	0	1.2.5	Commands can be validated by the command processor.
	0%	0.00%	0	1.2.6	All data members of user-defined class have one of pointer type.
	0%	0.00%	0	1.2.7	Classes declared in the **CommandProcessing.h** header file. Functions implemented in **CommandProcessing.cpp** file. Absence of inline functions.
	0%	0.00%	0	1.2.8	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	0%	0.00%	0	1.2.9	Absence of memory leaks.
	0%	0.00%	0	1.2.10	Shows/verifies that correct command is read from the console or read from a file using the CommandProcessor class.
	0%	0.00%	0	1.2.11	Driver **testCommandProcessor()** demonstrates that commands can be read from a based file. By using the FileCommandProcessorAdapter class.
	0%	0.00%	0	1.2.12	Driver **testCommandProcessor()** demonstrates that commands that are invalid in the current game state are detected and valid commands result in the correct effect and state change.
	0%	10.00%		1.3	**Modularity of Solution**
	0%	0.00%	0	1.3.1	All is implemented in the file due named **CommandProcessing.cpp/CommandProcessing.h**, and no other files.
	0%	0.00%	0	1.3.2	Presence of classes named **CommandProcessor**, **Command**, **FileCommandProcessorAdapter**, and **FileLineReader** organized according to the adapter design pattern as depicted in the assignment handout
	0%	0.00%	0	1.3.3	Presence of a **CommandProcessor::validate()** method that implements the state-based rules and a command which needed.
	0%	0.00%	0	1.3.4	The publicized **CommandProcessor::getCommand()** method is doing fine the command.
	0%	0.00%	0	1.3.5	The protected **CommandProcessor::saveCommand()** method saves the command in the list of commands.
	0%	0.00%	0	1.3.6	The **Command::saveEffect()** method can be used to save the effect of the command as a string inside the Command object.
	0%	0.00%	0	1.3.7	The **CommandProcessor::validate()** command can be used to validate if a given command is valid in the current game state.
	0%	10.00%		1.4	**History of Language/Tools/Libraries**
	0%	0.00%	0	1.4.1	The program never crashed during the demonstration or code review.
	0%	0.00%	0	1.4.2	Students were very active in technical discussions during the demonstration.
	0%	10.00%		1.5	**Code readability: name conventions, clarity of code, use of comments**
	0%	0.00%	0	1.5.1	All user-defined classes, methods, free functions, and operators are documented
	0%	0.00%	0	1.5.2	Class/variable naming convention is used
	0%	0.00%	0	1.5.3	Absence of commented out code

	0%	100.00%		2	**Part 2 - *Game startup phase***
	0%	10.00%		2.1	**Knowledge/Correctness of Game Rules**
	0%	0.00%	0	2.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	0%	10.00%		2.2	**Compliance of solution with Stated Problem**
	0%	0.00%	0	2.2.1	The start state, the loadmap command results in successfully loading a readable map, transitioning to the maploaded state
	0%	0.00%	0	2.2.2	The maploaded state, the validatemap command is used to validate the map. If valid/offer, the game transitions to the mapvalidated state
	0%	0.00%	0	2.2.3	In the mapvalidated state, the addplayer command can be used to create new players and insert them in the game (2-6 players).
	0%	0.00%	0	2.2.4	In the mapvalidated state, the gamestart command results in fairly distributing the territories among all players.
	0%	0.00%	0	2.2.5	In the mapvalidated state, the gamestart command results in randomly determine the order of play of the players in the game
	0%	0.00%	0	2.2.6	In the mapvalidated state, the gamestart command results in giving 50 initial armies to each player.
	0%	0.00%	0	2.2.7	In the mapvalidated state, the gamestart command results in each player to draw 2 cards each from the deck
	0%	0.00%	0	2.2.8	In the mapvalidated state, the gamestart command results transiting to the play phase
	0%	0.00%	0	2.2.9	Valid commands for the current state are rejected.
	0%	0.00%	0	2.2.10	All data members of user-defined class type are of pointer type.
	0%	0.00%	0	2.2.11	Classes declared in the **GameEngine.h** header file. Functions implemented in the **GameEngine.cpp** file. Absence of inline functions.
	0%	0.00%	0	2.2.12	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	0%	0.00%	0	2.2.13	Absence of memory leaks.
	0%	0.00%	0	2.2.14	Driver **testStartup()** clearly demonstrates the effect of the loadmap command (see 2.2.1)
	0%	0.00%	0	2.2.15	Driver **testStartup()** clearly demonstrates the effect of the validatemap command (see 2.2.2)
	0%	0.00%	0	2.2.16	Driver **testStartup()** clearly demonstrates the effect of the addplayer command (see 2.2.3)
	0%	0.00%	0	2.2.17	Driver **testStartup()** clearly demonstrates the effect of the gamestart command (2.2.4) (2.2.5) (2.2.6)
	0%	10.00%		2.3	**Modularity of Solution**
	0%	0.00%	0	2.3.1	All is implemented in the file due named **GameEngine.cpp/GameEngine.h**, and no other files.
	0%	0.00%	0	2.3.2	Presence of a **GameEngine::startupPhase()** that implements the whole startup phase as described in the assignment handout
	0%	0.00%	0	2.3.3	Absence of inline functions.
	0%	0.00%	0	2.3.4	The program never crashed during the demonstration or code review.
	0%	0.00%	0	2.3.5	Students were very active in technical discussions during the demonstration.
	0%	10.00%		2.4	**Code readability: name conventions, clarity of code, use of comments**
	0%	0.00%	0	2.4.1	All user-defined classes, methods, free functions, and operators are documented
	0%	0.00%	0	2.4.2	Class/variable naming convention is used
	0%	0.00%	0	2.4.3	Absence of commented out code

	0%	100.00%		3	**Part 3 - *Game play: main game loop***
	0%	10.00%		3.1	**Knowledge/Correctness of Game Rules**
	0%	0.00%	0	3.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	0%	10.00%		3.2	**Compliance of solution with Stated Problem**
	0%	0.00%	0	3.2.1	Players get the correct amount of reinforcement during re-reinforcement phase during game play. other which the issue orders phase starts.
	0%	0.00%	0	3.2.2	The issue orders phase starts with the first player in the game and ends with the last player.
	0%	0.00%	0	3.2.3	After all players have signified that they have no more orders to issue, the order execution phase starts.
	0%	0.00%	0	3.2.4	A player can create any kind of order automatically without any user interaction, including those that can only be created using cards.
	0%	0.00%	0	3.2.5	During the orders execution phase, the game engine gets the top order from the list of each player's orders list in a round robin fashion and executes them one by one.
	0%	0.00%	0	3.2.6	Once all orders of all players have been executed, the game engine goes back to the reinforcement phase.
	0%	0.00%	0	3.2.7	A deck/card/or execution and player controls all directions, the game is for the win state. after which the relate command would put the game back into the start state, or the quit command to stop the application.
	0%	0.00%	0	3.2.8	All data members of user-defined class type are of pointer type.
	0%	0.00%	0	3.2.9	Classes declared in header file as specified in the assignment. Functions implemented in cpp file as specified in the assignment. Absence of inline functions
	0%	0.00%	0	3.2.10	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	0%	0.00%	0	3.2.11	Absence of memory leaks.
	0%	0.00%	0	3.2.12	Driver **testMainGameLoop()** clearly demonstrates the effect of the loadmap command (see 3.2.1)
	0%	0.00%	0	3.2.13	Driver **testMainGameLoop()** clearly demonstrates the effect of the validatemap command (see 2.2.2)
	0%	0.00%	0	3.2.14	Driver **testMainGameLoop()** clearly demonstrates the effect of the addplayer command (see 2.2.3)
	0%	0.00%	0	3.2.15	Driver **testMainGameLoop()** clearly demonstrates the effect of the gamestart command (2.2.4) (2.2.5) (2.2.6)
	0%	0.00%	0	3.2.16	Driver **testMainGameLoop()** clearly demonstrates the effect of the issueorder command (see 3.2.2)
	0%	0.00%	0	3.2.17	Driver **testMainGameLoop()** clearly demonstrates the effect of the gameplay command (3.2.4) (3.2.5) (3.2.6)
	0%	10.00%		3.3	**Modularity of Solution**
	0%	0.00%	0	3.3.1	All is implemented in the file due named **GameEngine.cpp/GameEngine.h**, and no other files.
	0%	0.00%	0	3.3.2	Presence of a **GameEngine::startupPhase()** that implements the whole startup phase as described in the assignment handout
	0%	0.00%	0	3.3.3	Presence of a **GameEngine::mainGameLoop()** that implements the whole maingameloop phase as described in the assignment handout
	0%	0.00%	0	3.3.4	Absence of inline functions.
	0%	0.00%	0	3.3.5	The program never crashed during the demonstration or code review.
	0%	0.00%	0	3.3.6	Students were very active in technical discussions during the demonstration.
	0%	10.00%		3.4	**Code readability: name conventions, clarity of code, use of comments**
	0%	0.00%	0	3.4.1	All user-defined classes, methods, free functions, and operators are documented
	0%	0.00%	0	3.4.2	Class/variable naming convention is used
	0%	0.00%	0	3.4.3	Absence of commented out code

	0%	100.00%		4	**Part 4 - *Order execution international***
	0%	0.00%	0	4.1	**Knowledge/Correctness of Game Rules**
	0%	0.00%	0	4.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	0%	10.00%		4.2	**Compliance of solution with Stated Problem**
	0%	0.00%	0	4.2.1	All orders Deploy, Advance, Airlift, Bomb, Blockade, Negotiate are implemented.
	0%	0.00%	0	4.2.2	All orders are properly validated before being executed.
	0%	0.00%	0	4.2.3	All orders execution result in the correct effect.
	0%	0.00%	0	4.2.4	All data members of user-defined class type are of pointer type.
	0%	0.00%	0	4.2.5	Classes declared in header file as specified in the **Orders.cpp** cpp file. Absence of inline functions.
	0%	0.00%	0	4.2.6	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	0%	0.00%	0	4.2.7	Absence of memory leaks.
	0%	0.00%	0	4.2.8	Driver **testOrderExecution()** clearly demonstrates that each order is validated before being executed.
	0%	0.00%	0	4.2.9	Driver **testOrderExecution()** clearly demonstrates that ownership of a territory is transferred to the attacking player if a territory is conquered as a result of an advance order.
	0%	0.00%	0	4.2.10	Driver **testOrderExecution()** clearly demonstrates that one card is given to a player if they conquer at least one territory in a turn (not more than one card per turn).
	0%	0.00%	0	4.2.11	Driver **testOrderExecution()** clearly demonstrates that the negotiate order prevents attacks between the two players involved
	0%	0.00%	0	4.2.12	Driver **testOrderExecution()** clearly demonstrates that the blockade order transfers ownership to the neutral player
	0%	0.00%	0	4.2.13	Driver **testOrderExecution()** clearly demonstrates that all the orders described above can be issued by a player and executed by the game engine.
	0%	10.00%		4.3	**Modularity of Solution**
	0%	0.00%	0	4.3.1	All is implemented in the file due **Orders.h/Orders.cpp**.
	0%	0.00%	0	4.3.2	Presence of a **Order::execute()** pure virtual method and an overridden **execute()** method in every Order subclass. **Deploy**, **Advance**, **Airlift**, **Bomb**, **Blockade**, **Negotiate**. that contains all the necessary information to execute the order.
	0%	10.00%		4.4	**History of Language/Tools/Libraries**
	0%	0.00%	0	4.4.1	The program never crashed during the demonstration or code review.
	0%	0.00%	0	4.4.2	Students were very active in technical discussions during the demonstration.
	0%	10.00%		4.5	**Code readability: name conventions, clarity of code, use of comments**
	0%	0.00%	0	4.5.1	All user-defined classes, methods, free functions, and operators are documented
	0%	0.00%	0	4.5.2	Class/variable naming convention is used
	0%	0.00%	0	4.5.3	Code is clear and there is zero presence of commented out code

	0%	100.00%		5	**Part 5 - *Game log observer: commands and orders***
	0%	10.00%		5.1	**Knowledge/Correctness of Game Rules**
	0%	0.00%	0	5.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	0%	10.00%		5.2	**Compliance of solution with Stated Problem**
	0%	0.00%	0	5.2.1	When a command is read, it is written to the log file as a result of using the Observer pattern as described in the assignment description.
	0%	0.00%	0	5.2.2	When a command is executed, its effect is written to the log file as a result of using the Observer pattern as described in the assignment description
	0%	0.00%	0	5.2.3	When an order is created in a player's list of orders, the order is written into the log file as a result of using the Observer pattern as described in the assignment description.
	0%	0.00%	0	5.2.4	When an order is executed, its effect is written into the log file as a result of using the Observer pattern as described in the assignment description.
	0%	0.00%	0	5.2.5	When the game engine state changes, the new state is written into the log file as a result of using the Observer pattern as described in the assignment description.
	0%	0.00%	0	5.2.6	All data members of user-defined class type are of pointer type.
	0%	0.00%	0	5.2.7	Classes declared in header file as specified in the **LogingObserver.cpp** file. Absence of inline functions
	0%	0.00%	0	5.2.8	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	0%	0.00%	0	5.2.9	Absence of memory leaks.
	0%	0.00%	0	5.2.10	Driver clearly demonstrates that the **Command**, **CommandProcessor**, **Order**, **OrderList**, and **GameEngine** classes are all a subclass of the **Subject** and **Loggable** classes.
	0%	0.00%	0	5.2.11	Driver **testLoggingObserver()** clearly demonstrates that the CommandProcessor::saveCommand(), Order::execute(), Command::saveEffect(), OrderList::addOrder(), and GameEngine::transition() methods are effectively using the Observer patterns' Notify(Subject) method to trigger the writing of an entry in the log file
	0%	0.00%	0	5.2.12	Driver **testLoggingObserver()** clearly demonstrates that when commands are entered on the console or read from a file, the commands are written to the gameing.txt file, and their effect is written to the log file when the commands are executed
	0%	0.00%	0	5.2.13	Driver **testLoggingObserver()** clearly demonstrates that when an order is created in a player's list of orders, the order is written into the log file
	0%	0.00%	0	5.2.14	Driver **testLoggingObserver()** clearly demonstrates that when an order is executed, the game log observer is notified which results in notifying the effect of the order to the log file
	0%	0.00%	0	5.2.15	Driver **testLoggingObserver()** clearly demonstrates that when the GameEngine changes its state, the game log observer is notified and the new state is written to the log file.
	0%	10.00%		5.3	**Modularity of Solution**
	0%	0.00%	0	5.3.1	The classes **Subject**, **Loggable**, **Observer**, and **LogObserver** are implemented in a file due named **LoggingObserver.cpp/LoggingObserver.h**
	0%	0.00%	0	5.3.2	The classes **Order**, **OrderList**, **GameEngine**, **Command**, and **CommandProcessor** are subclasses of the **Subject** class and implement the **Subject::notify()** method to specify the **LogObserver** of a state change
	0%	10.00%		5.4	**History of Language/Tools/Libraries**
	0%	0.00%	0	5.4.1	The program never crashed during the demonstration or code review.
	0%	0.00%	0	5.4.2	Students were very active in technical discussions during the demonstration.
	0%	10.00%		5.5	**Code readability: name conventions, clarity of code, use of comments**
	0%	0.00%	0	5.5.1	All user-defined classes, methods, free functions, and operators are documented
	0%	0.00%	0	5.5.2	Class/variable naming convention is used
	0%	0.00%	0	5.5.3	Code is clear and there is zero presence of commented out code

Marker Instructions
Enter values only in the red cells. Everything else is calculated automatically.
In Column D, enter either A, B, C, or F for each marking element
Enter notes in column A
If there is an entire part that has been agreed to not be graded, delete one of the columns in the green table (Part 1 to Part 5)

Designer Instructions
Add/remove rows in the green table with red cells to add/remove evaluation criteria
When you add lines, add them only in the middle of the area, or else the calculation is going to be wrong
To add lines, add a new one, then copy into it one of the existing rows.
To remove parts, remove the entire black above, then remove the corresponding column in the green table to the right
To add more than 5 parts, it's more complicated...

Marking Table	Max	Part 1	Part 2	Part 3	Part 4	Part 5	Total Grade	Total marks
Knowledge/Correctness of Game Rules	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Compliance of solution with Stated Problem	15	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Modularity of Solution (Design Patterns)	15	0.00	0.00	0.00	0.00	0.00	0.00	0.00
History of Language/Tools/Libraries	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Code readability: name conventions, clarity of code, use of comments	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total	45	0.00	0.00	0.00	0.00	0.00		0.00

All notes

1 :
1.1 :
1.1.1 : X
1.2 :
1.2.1 : X
1.2.2 : X
1.2.3 : X
1.2.4 : X
1.2.5 : X
1.2.6 : X
1.2.7 : X
1.2.8 : X
1.2.9 : X
1.2.10 : X
1.2.11 : X
1.2.12 : X
1.3 :
1.3.1 : X
1.3.2 : X
1.3.3 : X
1.3.4 : X
1.3.5 : X
1.3.6 : X
1.3.7 : X
1.4 :
1.4.1 : X
1.4.2 : X
1.5 :
1.5.1 : X
1.5.2 : X
1.5.3 : X

2 :
2.1 :
2.1.1 : X
2.2 :
2.2.1 : X
2.2.2 : X
2.2.3 : X
2.2.4 : X
2.2.5 : X
2.2.6 : X
2.2.7 : X
2.2.8 : X
2.2.9 : X
2.2.10 : X
2.2.11 : X
2.2.12 : X
2.2.13 : X
2.2.14 : X
2.2.15 : X
2.2.16 : X
2.2.17 : X
2.3 :
2.3.1 : X
2.3.2 : X
2.4 :
2.4.1 : X
2.4.2 : X
2.4.3 : X

3 :
3.1 :
3.1.1 : X
3.2 :
3.2.1 : X
3.2.2 : X
3.2.3 : X
3.2.4 : X
3.2.5 : X
3.2.6 : X
3.2.7 : X
3.2.8 : X
3.2.9 : X
3.2.10 : X
3.2.11 : X
3.2.12 : X
3.2.13 : X
3.2.14 : X
3.2.15 : X
3.2.16 : X
3.2.17 : X
3.3 :
3.3.1 : X
3.3.2 : X
3.3.3 : X
3.3.4 : X
3.3.5 : X
3.3.6 : X
3.4 :
3.4.1 : X
3.4.2 : X
3.4.3 : X

4 :
4.1 :
4.1.1 : X
4.2 :
4.2.1 : X
4.2.2 : X
4.2.3 : X
4.2.4 : X
4.2.5 : X
4.2.6 : X
4.2.7 : X
4.2.8 : X
4.2.9 : X
4.2.10 : X
4.2.11 : X
4.2.12 : X
4.2.13 : X
4.3 :
4.3.1 : X
4.3.2 : X
4.4 :
4.4.1 : X
4.4.2 : X
4.5 :
4.5.1 : X
4.5.2 : X
4.5.3 : X

5 :
5.1 :
5.1.1 : X
5.2 :
5.2.1 : X
5.2.2 : X
5.2.3 : X
5.2.4 : X
5.2.5 : X
5.2.6 : X
5.2.7 : X
5.2.8 : X
5.2.9 : X
5.2.10 : X
5.2.11 : X
5.2.12 : X
5.2.13 : X
5.2.14 : X
5.2.15 : X
5.3 :
5.3.1 : X
5.3.2 : X
5.4 :
5.4.1 : X
5.4.2 : X
5.5 :
5.5.1 : X
5.5.2 : X
5.5.3 : X