Submitted by:	
Marker:	

Name	Value%	Ratio	Integer			
	0%	100.00%		0	**Part 1: Map**	1 :
	3%	30.00%		1.0	**Misconception/Correctness of Game Rules**	1.1 :
	3%	8.00%	30	1.1.1	Students can fully aware of the correct Map-zone game rules to implement during the presentation, and the code is implemented according to the rules of the game	1.1.1 : 0
	3%	60.00%		1.2	**Completeness of solution with Stated Problem**	1.2 :
	3%	8.00%	30	1.2.1	Map-class contains a collection of attributes that each tile Which	1.2.1 : 0
	3%	8.00%	30	1.2.2	Tile-class contains a collection of attributes	1.2.2 : 0
	3%	8.00%	30	1.2.3	"Instead of a Map, solutions) method	1.2.3 : 0
	3%	8.00%	30	1.2.4	Map is a collection of Territories	1.2.4 : 0
	3%	8.00%	30	1.2.5	Coordinates are a subset of the Territories in the puzzle	1.2.5 : 0
	3%	8.00%	30	1.2.6	Map can be made to represent any map configuration. Moderator can load a variety of map files	1.2.6 : 0
	3%	8.00%	30	1.2.7	Map is enable-tested using a graph data structure with Territories as nodes, and accesses by Territories as edges	1.2.7 : 0
	3%	8.00%	30	1.2.8	Map-class contains a collection of Territories	1.2.8 : 0
	3%	8.00%	30	1.2.9	Classes declared in header files have appropriate access specifiers. Absence of inline functions	1.2.9 : 0
	3%	8.00%	30	1.2.10	All classes implement a correct copy constructor, assignment operator, and stream insertion operator	1.2.10 : 0
	3%	8.00%	30	1.2.11	Absence of memory leaks	1.2.11 : 0
	3%	8.00%	30	1.2.12	Screenshots from the Map-validate() method clearly show that Tile map is a connected graph. If a subset is a connected subgraph. Each country belongs to one and only one continent	1.2.12 : 0
	3%	8.00%	30	1.2.13	Screenshots that narrates understanding how game state saved. Demonstration shows files can be read, resulting in the exact or very similar set of Territories maps	1.2.13 : 0
	3%	30.00%		1.3	**Modularity of Solution**	1.3 :
	3%	8.00%	30	1.3.1	Map is implemented in the file that named Map.cpp/Map.h, and no other files	1.3.1 : 0
	3%	8.00%	30	1.3.2	Map-class contains a collection of Territories, and Map-class "knows" of class Player to cater to Territory ownership	1.3.2 : 0
	3%	30.00%		1.4	**Mastery of Language Tools/Libraries**	1.4 :
	3%	8.00%	30	1.4.1	The program never crashed during the demonstration or code review	1.4.1 : 0
	3%	8.00%	30	1.4.2	Students were very clear in technical discussions during the demonstration	1.4.2 : 0
	3%	30.00%		1.5	**Code readability: name conventions, clarity of code, use of comments**	1.5 :
	3%	8.00%	30	1.5.1	All user-defined classes, methods, free functions, and constants are documented	1.5.1 : 0
	3%	8.00%	30	1.5.2	Clear indentation, no long lines/written is used	1.5.2 : 0
	3%	8.00%	30	1.5.3	Absence of commented-out code	1.5.3 : 0
	0%	100.00%		2	**Part 2 (Player)**	2 :
	3%	30.00%		2.1	**Misconception/Correctness of Game Rules**	2.1 :
	3%	8.00%	30	2.1.1	Students are fully aware of the correct Map-zone game rules to implement during the presentation, and the code is implemented according to the rules of the game	2.1.1 : 0
	3%	60.00%		2.2	**Completeness of solution with Stated Problem**	2.2 :
	3%	8.00%	30	2.2.1	A player includes a collection of garrisons to territories - using the territory class	2.2.1 : 0
	3%	8.00%	30	2.2.2	A player includes a hand of cards - using the hand class	2.2.2 : 0
	3%	8.00%	30	2.2.3	Player class includes a collection of Territories	2.2.3 : 0
	3%	8.00%	30	2.2.4	Player class includes attack and defend methods that rollout a list of values	2.2.4 : 0
	3%	8.00%	30	2.2.5	Player class includes an issueOrder method that creates an Order object and places it on the list of orders of the player	2.2.5 : 0
	3%	8.00%	30	2.2.6	All data members of user-defined class type are of pointer type	2.2.6 : 0
	3%	8.00%	30	2.2.7	Classes declared in header file: functions implemented in cpp file. Absence of inline functions	2.2.7 : 0
	3%	8.00%	30	2.2.8	All classes implement a correct copy constructor, assignment operator, and stream insertion operator	2.2.8 : 0
	3%	8.00%	30	2.2.9	Absence of memory leaks	2.2.9 : 0
	3%	8.00%	30	2.2.10	Screenshots that narrates that both get and set methods can be called and remove a card for instance	2.2.10 : 0
	3%	8.00%	30	2.2.11	Screenshots that demonstrates that issueOrder can be called that will result in an order to be placed in the list of orders of the player	2.2.11 : 0
	3%	8.00%	30	2.2.12	Screenshots that clearly demonstrates that a player knows a hand of cards	2.2.12 : 0
	3%	8.00%	30	2.2.13	Screenshots that clearly demonstrates that a player knows a list of owned territories	2.2.13 : 0
	3%	30.00%		2.3	**Modularity of Solution**	2.3 :
	3%	8.00%	30	2.3.1	Player is implemented in the file that named Player.cpp/Player.h	2.3.1 : 0
	3%	8.00%	30	2.3.2	Presence of Player class, usage of classes Territory, Hand, and Order List	2.3.2 : 0
	3%	30.00%		2.4	**Mastery of Language Tools/Libraries**	2.4 :
	3%	8.00%	30	2.4.1	The program never crashed during the demonstration or code review	2.4.1 : 0
	3%	8.00%	30	2.4.2	Students were very clear in technical discussions during the demonstration	2.4.2 : 0
	3%	30.00%		2.5	**Code readability: name conventions, clarity of code, use of comments**	2.5 :
	3%	8.00%	30	2.5.1	All user-defined classes, methods, free functions, and constants are documented	2.5.1 : 0
	3%	8.00%	30	2.5.2	Clear indentation, no long lines/written is used	2.5.2 : 0
	3%	8.00%	30	2.5.3	Absence of commented-out code	2.5.3 : 0
	0%	100.00%		3	**Part 3 (Order)**	3 :
	3%	30.00%		3.1	**Misconception/Correctness of Game Rules**	3.1 :
	3%	8.00%	30	3.1.1	Students are fully aware of the correct Map-zone game rules to implement during the presentation, and the code is implemented according to the rules of the game	3.1.1 : 0
	3%	60.00%		3.2	**Completeness of solution with Stated Problem**	3.2 :
	3%	8.00%	30	3.2.1	Deploy of any subclass can be created by the player and placed sequentially in the list of orders	3.2.1 : 0
	3%	8.00%	30	3.2.2	Orders can be reversed in the list using the reverse() method	3.2.2 : 0
	3%	8.00%	30	3.2.3	Orders have a removeOrder() method that removed method	3.2.3 : 0
	3%	8.00%	30	3.2.4	Orders have an execute() method	3.2.4 : 0
	3%	8.00%	30	3.2.5	All data members of user-defined class type are of pointer type	3.2.5 : 0
	3%	8.00%	30	3.2.6	Classes declared in header file: functions implemented in cpp file. Absence of inline functions	3.2.6 : 0
	3%	8.00%	30	3.2.7	All classes implement a correct copy constructor, assignment operator, and stream insertion operator	3.2.7 : 0
	3%	8.00%	30	3.2.8	Absence of memory leaks	3.2.8 : 0
	3%	8.00%	30	3.2.9	Screenshots that narrates that both get and set methods can be called and remove a card for instance	3.2.9 : 0
	3%	8.00%	30	3.2.10	Screenshots that clearly demonstrates that deploy creates an Order object and places it in the Players' list of orders, removes the card from the hand, and places it back in the Deck	3.2.10 : 0
	3%	30.00%		3.3	**Modularity of Solution**	3.3 :
	3%	8.00%	30	3.3.1	Order is implemented in the file that Orders.h/Orders.cpp	3.3.1 : 0
	3%	8.00%	30	3.3.2	Presence of OrderList, and Order classes. Presence of Deploy, Advance, Bomb, Blockade, Airlift, and Negotiate as a subclasses of the Order class	3.3.2 : 0
	3%	30.00%		3.4	**Mastery of Language Tools/Libraries**	3.4 :
	3%	8.00%	30	3.4.1	The program never crashed during the demonstration or code review	3.4.1 : 0
	3%	8.00%	30	3.4.2	Students were very clear in technical discussions during the demonstration	3.4.2 : 0
	3%	30.00%		3.5	**Code readability: name conventions, clarity of code, use of comments**	3.5 :
	3%	8.00%	30	3.5.1	All user-defined classes, methods, free functions, and constants are documented	3.5.1 : 0
	3%	8.00%	30	3.5.2	Clear indentation, no long lines/written is used	3.5.2 : 0
	3%	8.00%	30	3.5.3	Absence of commented-out code	3.5.3 : 0
	0%	100.00%		4	**Part 4 (Cards.h/cc/.h)**	4 :
	3%	30.00%		4.1	**Misconception/Correctness of Game Rules**	4.1 :
	3%	8.00%	30	4.1.1	Students are fully aware of the correct Map-zone game rules to implement during the presentation, and the code is implemented according to the rules of the game	4.1.1 : 0
	3%	60.00%		4.2	**Completeness of solution with Stated Problem**	4.2 :
	3%	8.00%	30	4.2.1	A Hand has a method draw() that removes the card from the Deck and returns that card	4.2.1 : 0
	3%	8.00%	30	4.2.2	A Hand class contains a collection of Cards	4.2.2 : 0
	3%	8.00%	30	4.2.3	A Card class has a play() method, which creates an Order object and places it in the Players' list of orders, removes the card from the hand, and places it back in the Deck	4.2.3 : 0
	3%	8.00%	30	4.2.4	All data members of user-defined class type are of pointer type	4.2.4 : 0
	3%	8.00%	30	4.2.5	Classes declared in header file: functions implemented in cpp file. Absence of inline functions	4.2.5 : 0
	3%	8.00%	30	4.2.6	All classes implement a correct copy constructor, assignment operator, and stream insertion operator	4.2.6 : 0
	3%	8.00%	30	4.2.7	Absence of memory leaks	4.2.7 : 0
	3%	8.00%	30	4.2.8	Screenshots that clearly demonstrates that draw() effectively removes the chosen card from the deck and returns a Card	4.2.8 : 0
	3%	8.00%	30	4.2.9	Screenshots that clearly demonstrates that play() creates an Order object and places it in the Players' list of orders, removes the card from the hand, and places it back in the Deck	4.2.9 : 0
	3%	30.00%		4.3	**Modularity of Solution**	4.3 :
	3%	8.00%	30	4.3.1	Hand is implemented in the file that Cards.h/Cards.cpp	4.3.1 : 0
	3%	8.00%	30	4.3.2	Presence of Card, Deck, and Hand classes. usage of classes Player, Order, and OrderList	4.3.2 : 0
	3%	30.00%		4.4	**Mastery of Language Tools/Libraries**	4.4 :
	3%	8.00%	30	4.4.1	The program never crashed during the demonstration or code review	4.4.1 : 0
	3%	8.00%	30	4.4.2	Students were very clear in technical discussions during the demonstration	4.4.2 : 0
	3%	30.00%		4.5	**Code readability: name conventions, clarity of code, use of comments**	4.5 :
	3%	8.00%	30	4.5.1	All user-defined classes, methods, free functions, or operators are documented	4.5.1 : 0
	3%	8.00%	30	4.5.2	Clear indentation, no long lines/written is used	4.5.2 : 0
	3%	8.00%	30	4.5.3	Code is clear and there is not a presence of commented-out code	4.5.3 : 0
	0%	100.00%		5	**Part 5 (Game Engine)**	5 :
	3%	30.00%		5.1	**Misconception/Correctness of Game Rules**	5.1 :
	3%	8.00%	30	5.1.1	Students are fully aware of the correct Map-zone game rules to implement during the presentation, and the code is implemented according to the rules of the game	5.1.1 : 0
	3%	60.00%		5.2	**Completeness of solution with Stated Problem**	5.2 :
	3%	8.00%	30	5.2.1	The code of this game is host in a .docx member of the GameEngine class	5.2.1 : 0
	3%	8.00%	30	5.2.2	Console input is used to get commands from the user	5.2.2 : 0
	3%	8.00%	30	5.2.3	Game class is a collection and has a startupPhase() mode to monitor state to changing the value held in the state variable	5.2.3 : 0
	3%	8.00%	30	5.2.4	Game class is a collection and has a mainGameLoop() mode to monitor state, the command collection, and an error message is output to the console	5.2.4 : 0
	3%	8.00%	30	5.2.5	The state transitions are studied written exactly as illustrated in the assignment handout	5.2.5 : 0
	3%	8.00%	30	5.2.6	All data members of user-defined class type are of pointer type	5.2.6 : 0
	3%	8.00%	30	5.2.7	Classes declared in header file: functions implemented in cpp file. Absence of inline functions	5.2.7 : 0
	3%	8.00%	30	5.2.8	All classes implement a correct copy constructor, assignment operator, and stream insertion operator	5.2.8 : 0
	3%	8.00%	30	5.2.9	Absence of memory leaks	5.2.9 : 0
	3%	30.00%		5.3	**Modularity of Solution**	5.3 :
	3%	8.00%	30	5.3.1	Implementation is in the documented GameEngine.cpp/GameEngine.h	5.3.1 : 0
	3%	8.00%	30	5.3.2	Presence of GameEngine class	5.3.2 : 0
	3%	30.00%		5.4	**Mastery of Language Tools/Libraries**	5.4 :
	3%	8.00%	30	5.4.1	The program never crashed during the demonstration or code review	5.4.1 : 0
	3%	8.00%	30	5.4.2	Students were very clear in technical discussions during the demonstration	5.4.2 : 0
	3%	30.00%		5.5	**Code readability: name conventions, clarity of code, use of comments**	5.5 :
	3%	8.00%	30	5.5.1	All user-defined classes, methods, free functions, or operators are documented	5.5.1 : 0
	3%	8.00%	30	5.5.2	Clear indentation, no long lines/written is used	5.5.2 : 0
	3%	8.00%	30	5.5.3	Code is clear and there is not a presence of commented-out code	5.5.3 : 0

Hidden comments:
Green values only in the red cells. Every Pt ng cell is calculated automatically
In Column G, enter either n, 0, C, or F for each marking element
Enter index in column A
If Part 0 is red, some part has been agreed to worth granted, choice one of the columns in the green table (Part 1 to Part 5)

Designer instructions:
Add element name rows with red cells to add/remove evaluation criteria
When you add rows, add these cells in the middle of the area, or else the calculations is going to be wrong
To add lines, add a new row, then copy into it one of the existing rows.
To remove parts, remove the white black above, then remove the corresponding column in the green table to the right
To add more than 5 parts, it's more complicated...

Grading rubric	Value	Part 1	Part 2	Part 3	Part 4	Part 5	Score info	Total marks
Misconception/Correctness of Game Rules	30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.0%	0.00
Completeness of solution with Stated Problem	60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.0%	0.00
Modularity of Solution	30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.0%	0.00
Mastery of Language Tools/Libraries	30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.0%	0.00
Code readability: name conventions, clarity of code, use of comments	30%	0.00%	0.00%	0.00%	0.00%	0.00%	0.0%	0.00
Total	210%	0.00%	0.00%	0.00%	0.00%	0.00%	0.0%	0.00

Full score
1 :
1.0 :
1.1.1 : X
1.1 :
1.2.1 : Y
1.2.0 : X
1.2.3 : Y
1.2.4 : X
1.2.5 : X
1.2.6 : X
1.2.7 : X
1.2.8 : Y
1.2.9 : X
1.2.10 : X
1.2.11 : X
1.2.12 : X
1.2.13 : Y
1.2 :
1.3.1 : X
1.3.0 : X
1.4 :
1.4.1 : X
1.4.2 : X
1.4 :
1.5.1 : X
1.5.2 : Y
1.5.3 : X
1 :
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 : Y
2.2.7 : X
2.2.8 : X
2.2.9 : X
2.2.10 : X
2.2.11 : Y
2.2.12 : X
2.2.13 : X
2.3 :
2.3.1 : X
2.3.2 : Y
2.4 :
2.4.1 : X
2.4.2 : X
2.5 :
2.5.1 : X
2.5.2 : X
2.5.3 : X
2 :
3 :
3.1 : 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
3.3 :
3.3.1 : X
3.3.2 : Y
3.4 :
3.4.1 : X
3.4.2 : X
3.5 :
3.5.1 : Y
3.5.2 : X
3.5.3 : Y
3 :
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
4.3 :
4.3.1 : X
4.3.2 : Y
4.4 :
4.4.1 : X
4.4.2 : X
4.5 :
4.5.1 : X
4.5.2 : Y
4.5.3 : X
4 :
5 :
5.1 :
5.1.1 : Y
5.2 :
5.2.1 : X
5.2.2 : X
5.2.3 : X
5.2.4 : Y
5.2.5 : X
5.2.6 : X
5.2.7 : X
5.2.8 : X
5.2.9 : X
5.3 :
5.3.1 : X
5.3.2 : Y
5.4 :
5.4.1 : X
5.4.2 : X
5.5 :
5.5.1 : X
5.5.2 : Y
5.5.3 : X
5 :