Submitted by:	
Marker:	
General Notes:	

Notes	%mark	ratio	letter		
	100%	**100.00%**		**1**	**Part 1 : Player strategy pattern**
	100%	10.00%		**1.1**	**Knowledge/Correctness of Game Rules**
	100%	5.00%	A	1.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	100%	5.00%	A	1.1.2	Code is implementing game mechanics that is fully according the Warzone game
	100%	60.00%		**1.2**	**Compliance of solution with Stated Problem**
	100%	4.62%	A	1.2.1	Human player: requires user interactions to make decisions
	100%	4.62%	A	1.2.2	Aggressive player: computer player that focuses on attack (deploys or advances armies on its strongest country, then always advances to enemy territories until it cannot do so anymore).
	100%	4.62%	A	1.2.3	Benevolent player: computer player that focuses on protecting its weak countries (deploys or advances armies on its weakest countries, never advances to enemy territories).
	100%	4.62%	A	1.2.4	Neutral player: computer player that never issues any order.
	100%	4.62%	A	1.2.5	Cheater player: computer player that automatically conquers all territories that are adjacent to its own territories (only once per turn).
	100%	4.62%	A	1.2.6	If a Neutral player is attacked, it becomes an Aggressive player.
	100%	4.62%	A	1.2.7	All data members of user-defined class type are of pointer type.
	100%	4.62%	A	1.2.8	Classes declared in header file. Functions implemented in cpp file. Absence of inline functions
	100%	4.62%	A	1.2.9	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	100%	4.62%	A	1.2.10	Absence of memory leaks.
	100%	4.62%	A	1.2.11	Driver that demonstrates that different players can be assigned different strategies that lead to different behavior using the Strategy design pattern.
	100%	4.62%	A	1.2.12	Driver that demonstrates that the strategy adopted by a player can be changed dynamically during play.
	100%	4.62%	A	1.2.13	Driver that demonstrates that the human player makes decisions according to user interaction, and computer players make decisions automatically, which are both implemented using the strategy pattern.
	100%	10.00%		**1.3**	**Modularity of Solution**
	100%	1.67%	A	1.3.1	All is implemented in the file duo named PlayerStrategies.cpp/PlayerStrategies.h and no other files.
	100%	1.67%	A	1.3.2	The Player class does not have subclasses that implement different behaviors
	100%	1.67%	A	1.3.3	Presence of a PlayerStrategy abstract class that is a superclass of all the player strategy behavioral classes.
	100%	1.67%	A	1.3.4	For each strategy as described above, you have a ConcreteStrategy class: HumanPlayerStrategy, AggressivePlayerStrategy, BenevolentPlayerStrategy, and NeutralPlayerStrategy that are subclasses of the PlayerStrategy class.
	100%	1.67%	A	1.3.5	Each of the ConcreteStrategy classes implement their own version of the issueOrder(), toAttack(), and toDefend() methods.
	100%	1.67%	A	1.3.6	The Player class contains a data member of type PlayerStrategy.
	100%	1.67%	A	1.3.7	The issueOrder(), toDefend(), and toAttack() methods of the player do not implement behavior and simply delegate their call to the corresponding methods in the PlayerStrategy member of the Player.
	100%	10.00%		**1.4**	**Mastery of Language/Tools/Libraries**
	100%	5.00%	A	1.4.1	The program never crashed during the demonstration or code review
	100%	5.00%	A	1.4.2	Students were very clear in technical discussions during the demonstration
	100%	10.00%		**1.5**	**Code readability: name conventions, clarity of code, use of comments**
	100%	5.00%	A	1.5.1	All user-defined classes, methods, free functions, and operators are documented
	100%	5.00%	A	1.5.2	Absence of commented-out code
	100%	**100.00%**		**2**	**Part 2 : Tournament mode**
	100%	10.00%		**2.1**	**Knowledge/Correctness of Game Rules**
	100%	5.00%	A	2.1.1	Students are fully aware of the correct Warzone game rules to implement during the presentation
	100%	5.00%	A	2.1.2	Code is implementing game mechanics that is fully according the Warzone game
	100%	60.00%		**2.2**	**Compliance of solution with Stated Problem**
	100%	5.00%	A	2.2.1	The tournament command can be entered either from the console or from a file.
	100%	5.00%	A	2.2.2	Invalid tournament commands are rejected.
	100%	5.00%	A	2.2.3	Map files are loaded and validated as a result of executing a tournament command.
	100%	5.00%	A	2.2.4	Players are created as a result of executing a tournament command.
	100%	5.00%	A	2.2.5	The specified number of games are executed with all the players specified as a result of executing a tournament command.
	100%	5.00%	A	2.2.6	The played games end as a draw after the specified number of turns have been played without a winner.
	100%	5.00%	A	2.2.7	The games in a tournament all play witout any user interaction.
	100%	5.00%	A	2.2.8	All data members of user-defined class type are of pointer type.
	100%	5.00%	A	2.2.9	Classes declared in header file. Functions implemented in cpp file. Absence of inline functions
	100%	5.00%	A	2.2.10	All classes implement a correct copy constructor, assignment operator, and stream insertion operator.
	100%	5.00%	A	2.2.11	Absence of memory leaks.
	100%	5.00%	A	2.2.12	Driver clearly demonstrates the tournament command can be processed and validated by the CommandProcessor, and executed by the GameEngine, resulting in a tournament being played as described above.
	100%	10.00%		**2.3**	**Modularity of Solution**
	100%	5.00%	A	2.3.1	The code for the processing of the tournament command must be implemented in the existing CommandProcessing.cpp/CommandProcessing.h file duo
	100%	5.00%	A	2.3.2	The code for the execution of the tournament must be implemented in the existing GameEngine.cpp/GameEngine.h file duo.
	100%	10.00%		**2.4**	**Mastery of Language/Tools/Libraries**
	100%	5.00%	A	2.4.1	The program never crashed during the demonstration or code review
	100%	5.00%	A	2.4.2	Students were very clear in technical discussions during the demonstration
	100%	10.00%		**2.5**	**Code readability: name conventions, clarity of code, use of comments**
	100%	5.00%	A	2.5.1	All user-defined classes, methods, free functions, and operators are documented
	100%	5.00%	A	2.5.2	Absence of commented-out code