

# COS214 Spice Girls

0.1

Generated by Doxygen 1.9.6



|  |          |
|--|----------|
| <b>1 Hierarchical Index</b>                  | <b>1</b> |
| 1.1 Class Hierarchy                          | 1        |
| <b>2 Class Index</b>                         | <b>3</b> |
| 2.1 Class List                               | 3        |
| <b>3 File Index</b>                          | <b>5</b> |
| 3.1 File List                                | 5        |
| <b>4 Class Documentation</b>                 | <b>7</b> |
| 4.1 AddOn Class Reference                    | 7        |
| 4.1.1 Detailed Description                   | 8        |
| 4.1.2 Constructor & Destructor Documentation | 8        |
| 4.1.2.1 AddOn()                              | 8        |
| 4.1.3 Member Function Documentation          | 8        |
| 4.1.3.1 clone()                              | 8        |
| 4.1.3.2 dealDamage()                         | 8        |
| 4.1.3.3 getEntity()                          | 9        |
| 4.1.3.4 getValue()                           | 9        |
| 4.1.3.5 setEntity()                          | 9        |
| 4.1.3.6 setValue()                           | 10       |
| 4.1.3.7 takeDamage()                         | 10       |
| 4.1.4 Member Data Documentation              | 11       |
| 4.1.4.1 entity                               | 11       |
| 4.1.4.2 value                                | 11       |
| 4.2 AerialType Class Reference               | 11       |
| 4.2.1 Detailed Description                   | 12       |
| 4.2.2 Constructor & Destructor Documentation | 12       |
| 4.2.2.1 AerialType()                         | 12       |
| 4.2.3 Member Function Documentation          | 12       |
| 4.2.3.1 clone()                              | 12       |
| 4.2.3.2 getTypeDesc()                        | 12       |
| 4.3 Aggressive Class Reference               | 13       |
| 4.3.1 Detailed Description                   | 13       |
| 4.3.2 Constructor & Destructor Documentation | 13       |
| 4.3.2.1 Aggressive()                         | 13       |
| 4.3.3 Member Function Documentation          | 13       |
| 4.3.3.1 clone()                              | 14       |
| 4.3.3.2 performStrat()                       | 14       |
| 4.4 Alliance Class Reference                 | 15       |
| 4.4.1 Detailed Description                   | 15       |
| 4.4.2 Constructor & Destructor Documentation | 15       |
| 4.4.2.1 Alliance() [1/2]                     | 16       |

|  |    |
|--|----|
| 4.4.2.2 Alliance() [2/2]                     | 16 |
| 4.4.2.3 ~Alliance()                          | 16 |
| 4.4.3 Member Function Documentation          | 17 |
| 4.4.3.1 addCountry()                         | 17 |
| 4.4.3.2 addFactory()                         | 17 |
| 4.4.3.3 addReserveEntity()                   | 18 |
| 4.4.3.4 clone()                              | 18 |
| 4.4.3.5 considerPeace()                      | 19 |
| 4.4.3.6 getActive()                          | 19 |
| 4.4.3.7 getID()                              | 20 |
| 4.4.3.8 getReserveEntities()                 | 20 |
| 4.4.3.9 numRemainingEntities()               | 21 |
| 4.4.3.10 offerPeace()                        | 21 |
| 4.4.3.11 runFactories()                      | 21 |
| 4.4.3.12 setActiveStatus()                   | 22 |
| 4.4.3.13 setNegotiator()                     | 22 |
| 4.4.3.14 surrender()                         | 23 |
| 4.5 AquaticType Class Reference              | 23 |
| 4.5.1 Detailed Description                   | 24 |
| 4.5.2 Constructor & Destructor Documentation | 24 |
| 4.5.2.1 AquaticType()                        | 24 |
| 4.5.3 Member Function Documentation          | 24 |
| 4.5.3.1 clone()                              | 24 |
| 4.5.3.2 getTypeDesc()                        | 24 |
| 4.6 Area Class Reference                     | 25 |
| 4.6.1 Detailed Description                   | 25 |
| 4.6.2 Constructor & Destructor Documentation | 25 |
| 4.6.2.1 Area()                               | 25 |
| 4.6.2.2 ~Area()                              | 26 |
| 4.6.3 Member Function Documentation          | 26 |
| 4.6.3.1 addGeneral()                         | 26 |
| 4.6.3.2 clone()                              | 26 |
| 4.6.3.3 getAreaName()                        | 26 |
| 4.6.3.4 isKeyPoint()                         | 27 |
| 4.6.3.5 simulateBattle()                     | 27 |
| 4.7 Armour Class Reference                   | 27 |
| 4.7.1 Detailed Description                   | 28 |
| 4.7.2 Constructor & Destructor Documentation | 28 |
| 4.7.2.1 Armour()                             | 28 |
| 4.7.3 Member Function Documentation          | 28 |
| 4.7.3.1 clone()                              | 28 |
| 4.7.3.2 dealDamage()                         | 29 |

|   |    |
|---|----|
| 4.7.3.3 takeDamage()                          | 29 |
| 4.8 Cloudy Class Reference                    | 30 |
| 4.8.1 Detailed Description                    | 30 |
| 4.8.2 Constructor & Destructor Documentation  | 31 |
| 4.8.2.1 Cloudy()                              | 31 |
| 4.8.3 Member Function Documentation           | 31 |
| 4.8.3.1 clone()                               | 31 |
| 4.8.3.2 getWeather()                          | 31 |
| 4.8.3.3 handleChange()                        | 32 |
| 4.9 Country Class Reference                   | 32 |
| 4.9.1 Detailed Description                    | 33 |
| 4.9.2 Constructor & Destructor Documentation  | 33 |
| 4.9.2.1 Country()                             | 33 |
| 4.9.3 Member Function Documentation           | 33 |
| 4.9.3.1 clone()                               | 33 |
| 4.9.3.2 getID()                               | 34 |
| 4.9.3.3 getName()                             | 34 |
| 4.10 Defensive Class Reference                | 34 |
| 4.10.1 Detailed Description                   | 35 |
| 4.10.2 Constructor & Destructor Documentation | 35 |
| 4.10.2.1 Defensive()                          | 35 |
| 4.10.3 Member Function Documentation          | 35 |
| 4.10.3.1 clone()                              | 35 |
| 4.10.3.2 performStrat()                       | 35 |
| 4.11 EasySetup Class Reference                | 36 |
| 4.11.1 Detailed Description                   | 36 |
| 4.11.2 Constructor & Destructor Documentation | 36 |
| 4.11.2.1 EasySetup()                          | 36 |
| 4.11.3 Member Function Documentation          | 37 |
| 4.11.3.1 loadPrevSave()                       | 37 |
| 4.11.3.2 loadSpecificSave()                   | 37 |
| 4.11.3.3 runSimulation()                      | 37 |
| 4.11.3.4 saveSimulationSetup()                | 38 |
| 4.11.3.5 setupSimulation()                    | 38 |
| 4.12 Entity Class Reference                   | 41 |
| 4.12.1 Detailed Description                   | 42 |
| 4.12.2 Constructor & Destructor Documentation | 42 |
| 4.12.2.1 Entity() [1/2]                       | 42 |
| 4.12.2.2 Entity() [2/2]                       | 42 |
| 4.12.3 Member Function Documentation          | 43 |
| 4.12.3.1 clone()                              | 43 |
| 4.12.3.2 dealDamage()                         | 43 |

|           |  |    |
|-----------|--|----|
| 4.12.3.3  | <a href="#">getAlliance()</a>                              | 43 |
| 4.12.3.4  | <a href="#">getDamage()</a>                                | 44 |
| 4.12.3.5  | <a href="#">getHealth()</a>                                | 44 |
| 4.12.3.6  | <a href="#">getType()</a>                                  | 44 |
| 4.12.3.7  | <a href="#">setAlliance()</a>                              | 45 |
| 4.12.3.8  | <a href="#">setDamage()</a>                                | 45 |
| 4.12.3.9  | <a href="#">setHealth()</a>                                | 46 |
| 4.12.3.10 | <a href="#">setType()</a>                                  | 46 |
| 4.12.3.11 | <a href="#">takeDamage()</a>                               | 47 |
| 4.13      | <a href="#">Factory Class Reference</a>                    | 47 |
| 4.13.1    | <a href="#">Detailed Description</a>                       | 48 |
| 4.13.2    | <a href="#">Constructor &amp; Destructor Documentation</a> | 48 |
| 4.13.2.1  | <a href="#">Factory()</a>                                  | 48 |
| 4.13.2.2  | <a href="#">~Factory()</a>                                 | 48 |
| 4.13.3    | <a href="#">Member Function Documentation</a>              | 48 |
| 4.13.3.1  | <a href="#">clone()</a>                                    | 49 |
| 4.13.3.2  | <a href="#">createEntity()</a>                             | 49 |
| 4.13.3.3  | <a href="#">getAddOn()</a>                                 | 49 |
| 4.13.3.4  | <a href="#">getType()</a>                                  | 49 |
| 4.13.3.5  | <a href="#">setAddOns()</a>                                | 50 |
| 4.13.3.6  | <a href="#">setType()</a>                                  | 50 |
| 4.14      | <a href="#">General Class Reference</a>                    | 51 |
| 4.14.1    | <a href="#">Detailed Description</a>                       | 51 |
| 4.14.2    | <a href="#">Constructor &amp; Destructor Documentation</a> | 51 |
| 4.14.2.1  | <a href="#">General()</a>                                  | 51 |
| 4.14.3    | <a href="#">Member Function Documentation</a>              | 52 |
| 4.14.3.1  | <a href="#">getAlliance()</a>                              | 52 |
| 4.14.3.2  | <a href="#">initiateStrategy()</a>                         | 52 |
| 4.14.3.3  | <a href="#">setStrategy()</a>                              | 53 |
| 4.15      | <a href="#">KeyPoint Class Reference</a>                   | 54 |
| 4.15.1    | <a href="#">Detailed Description</a>                       | 54 |
| 4.15.2    | <a href="#">Constructor &amp; Destructor Documentation</a> | 55 |
| 4.15.2.1  | <a href="#">KeyPoint()</a> [1/2]                           | 55 |
| 4.15.2.2  | <a href="#">KeyPoint()</a> [2/2]                           | 55 |
| 4.15.2.3  | <a href="#">~KeyPoint()</a>                                | 55 |
| 4.15.3    | <a href="#">Member Function Documentation</a>              | 56 |
| 4.15.3.1  | <a href="#">addEntity()</a>                                | 56 |
| 4.15.3.2  | <a href="#">addGeneral()</a>                               | 56 |
| 4.15.3.3  | <a href="#">changeWeather()</a>                            | 57 |
| 4.15.3.4  | <a href="#">clearBattlefield()</a>                         | 57 |
| 4.15.3.5  | <a href="#">clone()</a>                                    | 58 |
| 4.15.3.6  | <a href="#">getWeather()</a>                               | 58 |

|   |    |
|---|----|
| 4.15.3.7 isKeyPoint()                         | 59 |
| 4.15.3.8 moveEntitiesInto()                   | 59 |
| 4.15.3.9 moveEntitiesOutOf()                  | 60 |
| 4.15.3.10 removeGeneral()                     | 60 |
| 4.15.3.11 setWeather()                        | 61 |
| 4.15.3.12 simulateBattle()                    | 61 |
| 4.16 Negotiator Class Reference               | 62 |
| 4.16.1 Detailed Description                   | 63 |
| 4.16.2 Constructor & Destructor Documentation | 63 |
| 4.16.2.1 Negotiator()                         | 63 |
| 4.16.2.2 ~Negotiator()                        | 63 |
| 4.16.3 Member Function Documentation          | 63 |
| 4.16.3.1 addAlliance()                        | 63 |
| 4.16.3.2 getNumAlliances()                    | 64 |
| 4.16.3.3 removeAlliance()                     | 64 |
| 4.16.3.4 sendPeace()                          | 64 |
| 4.17 Passive Class Reference                  | 65 |
| 4.17.1 Detailed Description                   | 65 |
| 4.17.2 Constructor & Destructor Documentation | 66 |
| 4.17.2.1 Passive()                            | 66 |
| 4.17.3 Member Function Documentation          | 66 |
| 4.17.3.1 clone()                              | 66 |
| 4.17.3.2 performStrat()                       | 66 |
| 4.18 Personnel Class Reference                | 67 |
| 4.18.1 Detailed Description                   | 67 |
| 4.18.2 Constructor & Destructor Documentation | 68 |
| 4.18.2.1 Personnel()                          | 68 |
| 4.18.3 Member Function Documentation          | 68 |
| 4.18.3.1 clone()                              | 68 |
| 4.18.3.2 dealDamage()                         | 69 |
| 4.18.3.3 takeDamage()                         | 69 |
| 4.19 PersonnelFactory Class Reference         | 70 |
| 4.19.1 Detailed Description                   | 70 |
| 4.19.2 Constructor & Destructor Documentation | 71 |
| 4.19.2.1 PersonnelFactory()                   | 71 |
| 4.19.3 Member Function Documentation          | 71 |
| 4.19.3.1 clone()                              | 71 |
| 4.19.3.2 createEntity()                       | 72 |
| 4.20 Piercing Class Reference                 | 72 |
| 4.20.1 Detailed Description                   | 73 |
| 4.20.2 Constructor & Destructor Documentation | 73 |
| 4.20.2.1 Piercing()                           | 73 |

|   |    |
|---|----|
| 4.20.3 Member Function Documentation          | 73 |
| 4.20.3.1 clone()                              | 73 |
| 4.20.3.2 dealDamage()                         | 74 |
| 4.20.3.3 takeDamage()                         | 75 |
| 4.21 Rainy Class Reference                    | 75 |
| 4.21.1 Detailed Description                   | 76 |
| 4.21.2 Constructor & Destructor Documentation | 76 |
| 4.21.2.1 Rainy()                              | 76 |
| 4.21.3 Member Function Documentation          | 76 |
| 4.21.3.1 clone()                              | 76 |
| 4.21.3.2 getWeather()                         | 77 |
| 4.21.3.3 handleChange()                       | 77 |
| 4.22 RoundStats Class Reference               | 78 |
| 4.22.1 Detailed Description                   | 78 |
| 4.22.2 Member Function Documentation          | 78 |
| 4.22.2.1 clearStats()                         | 78 |
| 4.22.2.2 toString()                           | 79 |
| 4.22.3 Member Data Documentation              | 79 |
| 4.22.3.1 damageDone                           | 79 |
| 4.22.3.2 entityMovementInformation            | 79 |
| 4.22.3.3 keyPointInformation                  | 79 |
| 4.22.3.4 numEntitiesCreated                   | 79 |
| 4.22.3.5 numEntitiesDestroyed                 | 80 |
| 4.23 SaveArchive Class Reference              | 80 |
| 4.23.1 Detailed Description                   | 80 |
| 4.23.2 Constructor & Destructor Documentation | 80 |
| 4.23.2.1 SaveArchive()                        | 81 |
| 4.23.3 Member Function Documentation          | 81 |
| 4.23.3.1 addNewSave()                         | 81 |
| 4.23.3.2 clearSaveList()                      | 81 |
| 4.23.3.3 deleteSave()                         | 82 |
| 4.23.3.4 getLastSave()                        | 83 |
| 4.23.3.5 getSave()                            | 83 |
| 4.24 Strategy Class Reference                 | 84 |
| 4.24.1 Detailed Description                   | 84 |
| 4.24.2 Constructor & Destructor Documentation | 85 |
| 4.24.2.1 Strategy()                           | 85 |
| 4.24.2.2 ~Strategy()                          | 85 |
| 4.24.3 Member Function Documentation          | 85 |
| 4.24.3.1 clone()                              | 85 |
| 4.24.3.2 performStrat()                       | 85 |
| 4.24.4 Member Data Documentation              | 85 |



|   |    |
|---|----|
| 4.24.4.1 strategy . . . . .                             | 86 |
| 4.25 Sunny Class Reference . . . . .                    | 86 |
| 4.25.1 Detailed Description . . . . .                   | 86 |
| 4.25.2 Constructor & Destructor Documentation . . . . . | 86 |
| 4.25.2.1 Sunny() . . . . .                              | 87 |
| 4.25.3 Member Function Documentation . . . . .          | 87 |
| 4.25.3.1 clone() . . . . .                              | 87 |
| 4.25.3.2 getWeather() . . . . .                         | 87 |
| 4.25.3.3 handleChange() . . . . .                       | 88 |
| 4.26 Support Class Reference . . . . .                  | 88 |
| 4.26.1 Detailed Description . . . . .                   | 89 |
| 4.26.2 Constructor & Destructor Documentation . . . . . | 89 |
| 4.26.2.1 Support() . . . . .                            | 89 |
| 4.26.3 Member Function Documentation . . . . .          | 89 |
| 4.26.3.1 clone() . . . . .                              | 90 |
| 4.26.3.2 dealDamage() . . . . .                         | 90 |
| 4.26.3.3 takeDamage() . . . . .                         | 91 |
| 4.27 SupportFactory Class Reference . . . . .           | 91 |
| 4.27.1 Detailed Description . . . . .                   | 92 |
| 4.27.2 Constructor & Destructor Documentation . . . . . | 92 |
| 4.27.2.1 SupportFactory() . . . . .                     | 92 |
| 4.27.3 Member Function Documentation . . . . .          | 92 |
| 4.27.3.1 clone() . . . . .                              | 92 |
| 4.27.3.2 createEntity() . . . . .                       | 93 |
| 4.28 TerrainType Class Reference . . . . .              | 94 |
| 4.28.1 Detailed Description . . . . .                   | 94 |
| 4.28.2 Constructor & Destructor Documentation . . . . . | 94 |
| 4.28.2.1 TerrainType() . . . . .                        | 94 |
| 4.28.3 Member Function Documentation . . . . .          | 94 |
| 4.28.3.1 clone() . . . . .                              | 95 |
| 4.28.3.2 getTypeDesc() . . . . .                        | 95 |
| 4.29 Type Class Reference . . . . .                     | 95 |
| 4.29.1 Detailed Description . . . . .                   | 96 |
| 4.29.2 Constructor & Destructor Documentation . . . . . | 96 |
| 4.29.2.1 Type() . . . . .                               | 96 |
| 4.29.3 Member Function Documentation . . . . .          | 96 |
| 4.29.3.1 getTypeDesc() . . . . .                        | 96 |
| 4.30 Vehicle Class Reference . . . . .                  | 97 |
| 4.30.1 Detailed Description . . . . .                   | 97 |
| 4.30.2 Constructor & Destructor Documentation . . . . . | 97 |
| 4.30.2.1 Vehicle() . . . . .                            | 97 |
| 4.30.3 Member Function Documentation . . . . .          | 98 |

|   |     |
|---|-----|
| 4.30.3.1 clone()                                  | 98  |
| 4.30.3.2 dealDamage()                             | 98  |
| 4.30.3.3 takeDamage()                             | 99  |
| 4.31 VehicleFactory Class Reference               | 100 |
| 4.31.1 Detailed Description                       | 100 |
| 4.31.2 Constructor & Destructor Documentation     | 100 |
| 4.31.2.1 VehicleFactory()                         | 100 |
| 4.31.3 Member Function Documentation              | 101 |
| 4.31.3.1 clone()                                  | 101 |
| 4.31.3.2 createEntity()                           | 101 |
| 4.32 WarEngine Class Reference                    | 102 |
| 4.32.1 Detailed Description                       | 103 |
| 4.32.2 Constructor & Destructor Documentation     | 103 |
| 4.32.2.1 WarEngine() [1/2]                        | 103 |
| 4.32.2.2 WarEngine() [2/2]                        | 103 |
| 4.32.2.3 ~WarEngine()                             | 103 |
| 4.32.3 Member Function Documentation              | 104 |
| 4.32.3.1 addAlliance()                            | 104 |
| 4.32.3.2 getInstance()                            | 104 |
| 4.32.3.3 loadSave()                               | 104 |
| 4.32.3.4 operator=()                              | 105 |
| 4.32.3.5 saveState()                              | 105 |
| 4.32.3.6 setWarTheatre()                          | 105 |
| 4.32.3.7 simulate()                               | 106 |
| 4.33 WarEngineMemento Class Reference             | 107 |
| 4.33.1 Detailed Description                       | 107 |
| 4.33.2 Friends And Related Function Documentation | 107 |
| 4.33.2.1 WarEngine                                | 107 |
| 4.34 WarEngineState Class Reference               | 107 |
| 4.34.1 Detailed Description                       | 107 |
| 4.34.2 Friends And Related Function Documentation | 108 |
| 4.34.2.1 WarEngine                                | 108 |
| 4.35 WarTheatre Class Reference                   | 108 |
| 4.35.1 Detailed Description                       | 108 |
| 4.35.2 Constructor & Destructor Documentation     | 109 |
| 4.35.2.1 WarTheatre()                             | 109 |
| 4.35.2.2 ~WarTheatre()                            | 109 |
| 4.35.3 Member Function Documentation              | 109 |
| 4.35.3.1 addArea()                                | 109 |
| 4.35.3.2 addGeneral()                             | 110 |
| 4.35.3.3 clone()                                  | 110 |
| 4.35.3.4 isKeyPoint()                             | 111 |

|   |            |
|---|------------|
| 4.35.3.5 simulateBattle()                     | 111        |
| 4.36 Weather Class Reference                  | 112        |
| 4.36.1 Detailed Description                   | 113        |
| 4.36.2 Constructor & Destructor Documentation | 113        |
| 4.36.2.1 Weather()                            | 113        |
| 4.36.2.2 ~Weather()                           | 113        |
| 4.36.3 Member Function Documentation          | 113        |
| 4.36.3.1 clone()                              | 113        |
| 4.36.3.2 getMultiplier()                      | 114        |
| 4.36.3.3 getWeather()                         | 114        |
| 4.36.3.4 handleChange()                       | 114        |
| 4.36.4 Member Data Documentation              | 114        |
| 4.36.4.1 multiplier                           | 114        |
| <b>5 File Documentation</b>                   | <b>115</b> |
| 5.1 AddOn.cpp                                 | 115        |
| 5.2 AddOn.h                                   | 115        |
| 5.3 AddOnTest.h                               | 116        |
| 5.4 AerialType.cpp                            | 118        |
| 5.5 AerialType.h                              | 118        |
| 5.6 Aggressive.cpp                            | 119        |
| 5.7 Aggressive.h                              | 119        |
| 5.8 Alliance.cpp                              | 119        |
| 5.9 Alliance.h                                | 120        |
| 5.10 AquaticType.cpp                          | 121        |
| 5.11 AquaticType.h                            | 121        |
| 5.12 Area.cpp                                 | 122        |
| 5.13 Area.h                                   | 122        |
| 5.14 Armour.cpp                               | 122        |
| 5.15 Armour.h                                 | 123        |
| 5.16 Cloudy.cpp                               | 123        |
| 5.17 Cloudy.h                                 | 123        |
| 5.18 Country.cpp                              | 124        |
| 5.19 Country.h                                | 124        |
| 5.20 Defensive.cpp                            | 124        |
| 5.21 Defensive.h                              | 124        |
| 5.22 EasySetup.cpp                            | 125        |
| 5.23 EasySetup.h                              | 128        |
| 5.24 Entity.cpp                               | 129        |
| 5.25 Entity.h                                 | 129        |
| 5.26 Factory.cpp                              | 130        |
| 5.27 Factory.h                                | 130        |

|                           |     |
|---------------------------|-----|
| 5.28 General.cpp          | 131 |
| 5.29 General.h            | 131 |
| 5.30 KeyPoint.cpp         | 132 |
| 5.31 KeyPoint.h           | 134 |
| 5.32 Negotiator.cpp       | 134 |
| 5.33 Negotiator.h         | 135 |
| 5.34 NegotiatorTest.h     | 135 |
| 5.35 Passive.cpp          | 136 |
| 5.36 Passive.h            | 136 |
| 5.37 Personnel.cpp        | 136 |
| 5.38 Personnel.h          | 137 |
| 5.39 PersonnelFactory.cpp | 137 |
| 5.40 PersonnelFactory.h   | 137 |
| 5.41 Piercing.cpp         | 138 |
| 5.42 Piercing.h           | 138 |
| 5.43 Rainy.cpp            | 138 |
| 5.44 Rainy.h              | 139 |
| 5.45 RoundStats.cpp       | 139 |
| 5.46 RoundStats.h         | 139 |
| 5.47 SaveArchive.cpp      | 140 |
| 5.48 SaveArchive.h        | 140 |
| 5.49 Strategy.cpp         | 141 |
| 5.50 Strategy.h           | 141 |
| 5.51 Sunny.cpp            | 141 |
| 5.52 Sunny.h              | 142 |
| 5.53 Support.cpp          | 142 |
| 5.54 Support.h            | 142 |
| 5.55 SupportFactory.cpp   | 142 |
| 5.56 SupportFactory.h     | 143 |
| 5.57 TerrainType.cpp      | 143 |
| 5.58 TerrainType.h        | 143 |
| 5.59 testmain.cpp         | 144 |
| 5.60 Type.cpp             | 144 |
| 5.61 Type.h               | 144 |
| 5.62 Vehicle.cpp          | 145 |
| 5.63 Vehicle.h            | 145 |
| 5.64 VehicleFactory.cpp   | 145 |
| 5.65 VehicleFactory.h     | 145 |
| 5.66 WarEngine.cpp        | 146 |
| 5.67 WarEngine.h          | 146 |
| 5.68 WarEngineMemento.cpp | 147 |
| 5.69 WarEngineMemento.h   | 147 |

---

|                                   |            |
|-----------------------------------|------------|
| 5.70 WarEngineState.cpp . . . . . | 148        |
| 5.71 WarEngineState.h . . . . .   | 148        |
| 5.72 WarTheatre.cpp . . . . .     | 149        |
| 5.73 WarTheatre.h . . . . .       | 149        |
| 5.74 Weather.cpp . . . . .        | 150        |
| 5.75 Weather.h . . . . .          | 150        |
| <b>Index</b>                      | <b>151</b> |



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

|                            |     |
|----------------------------|-----|
| Alliance . . . . .         | 15  |
| Area . . . . .             | 25  |
| KeyPoint . . . . .         | 54  |
| WarTheatre . . . . .       | 108 |
| Country . . . . .          | 32  |
| EasySetup . . . . .        | 36  |
| Entity . . . . .           | 41  |
| AddOn . . . . .            | 7   |
| Armour . . . . .           | 27  |
| Piercing . . . . .         | 72  |
| Personnel . . . . .        | 67  |
| Support . . . . .          | 88  |
| Vehicle . . . . .          | 97  |
| Factory . . . . .          | 47  |
| PersonnelFactory . . . . . | 70  |
| SupportFactory . . . . .   | 91  |
| VehicleFactory . . . . .   | 100 |
| General . . . . .          | 51  |
| Negotiator . . . . .       | 62  |
| RoundStats . . . . .       | 78  |
| SaveArchive . . . . .      | 80  |
| Strategy . . . . .         | 84  |
| Aggressive . . . . .       | 13  |
| Defensive . . . . .        | 34  |
| Passive . . . . .          | 65  |
| Type . . . . .             | 95  |
| AerialType . . . . .       | 11  |
| AquaticType . . . . .      | 23  |
| TerrainType . . . . .      | 94  |
| WarEngine . . . . .        | 102 |
| WarEngineMemento . . . . . | 107 |
| WarEngineState . . . . .   | 107 |
| Weather . . . . .          | 112 |
| Cloudy . . . . .           | 30  |
| Rainy . . . . .            | 75  |
| Sunny . . . . .            | 86  |





## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

|   |       |    |
|---|-------|----|
| AddOn   |       |    |
| AddOn   | class | 7  |
| AerialType  |       |    |
| AerialType  | class | 11 |
| Aggressive  |       | 13 |
| Alliance  |       | 15 |
| AquaticType   |       |    |
| AquaticType   | class | 23 |
| Area  |       | 25 |
| Armour  |       |    |
| Armour  | class | 27 |
| Cloudy  |       | 30 |
| Country   |       | 32 |
| Defensive   |       | 34 |
| EasySetup   |       | 36 |
| Entity  |       |    |
| Entity  | class | 41 |
| Factory   |       |    |
| Factory   | class | 47 |
| General   |       | 51 |
| KeyPoint  |       |    |
| Keypoint  | class | 54 |
| Negotiator  |       | 62 |
| Passive   |       | 65 |
| Personnel   |       |    |
| Personnel   | class | 67 |
| PersonnelFactory                                      |       |    |
| PersonnelFactory                                      | class | 70 |
| Piercing  |       |    |
| Piercing  | class | 72 |
| Rainy   |       | 75 |
| RoundStats  |       | 78 |
| SaveArchive   |       |    |
| Stores a list of mementos containing simulation state |       | 80 |
| Strategy  |       | 84 |

|  |     |
|--|-----|
| Sunny . . . . .  | 86  |
| Support  |     |
| Support class . . . . .  | 88  |
| SupportFactory   |     |
| SupportFactory class . . . . .                                 | 91  |
| TerrainType  |     |
| TerrainType class . . . . .                                    | 94  |
| Type   |     |
| Type class . . . . .   | 95  |
| Vehicle  |     |
| Vehicle class . . . . .  | 97  |
| VehicleFactory   |     |
| VehicleFactory class . . . . .                                 | 100 |
| WarEngine . . . . .  | 102 |
| WarEngineMemento . . . . .                                     | 107 |
| WarEngineState   |     |
| Class for storing current state of entire simulation . . . . . | 107 |
| WarTheatre . . . . .   | 108 |
| Weather . . . . .  | 112 |

## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

|                                  |    |
|----------------------------------|----|
| <a href="#">AddOn.cpp</a>        | ?? |
| <a href="#">AddOn.h</a>          | ?? |
| <a href="#">AddOnTest.h</a>      | ?? |
| <a href="#">AerialType.cpp</a>   | ?? |
| <a href="#">AerialType.h</a>     | ?? |
| <a href="#">Aggressive.cpp</a>   | ?? |
| <a href="#">Aggressive.h</a>     | ?? |
| <a href="#">Alliance.cpp</a>     | ?? |
| <a href="#">Alliance.h</a>       | ?? |
| <a href="#">AquaticType.cpp</a>  | ?? |
| <a href="#">AquaticType.h</a>    | ?? |
| <a href="#">Area.cpp</a>         | ?? |
| <a href="#">Area.h</a>           | ?? |
| <a href="#">Armour.cpp</a>       | ?? |
| <a href="#">Armour.h</a>         | ?? |
| <a href="#">Cloudy.cpp</a>       | ?? |
| <a href="#">Cloudy.h</a>         | ?? |
| <a href="#">Country.cpp</a>      | ?? |
| <a href="#">Country.h</a>        | ?? |
| <a href="#">Defensive.cpp</a>    | ?? |
| <a href="#">Defensive.h</a>      | ?? |
| <a href="#">EasySetup.cpp</a>    | ?? |
| <a href="#">EasySetup.h</a>      | ?? |
| <a href="#">Entity.cpp</a>       | ?? |
| <a href="#">Entity.h</a>         | ?? |
| <a href="#">Factory.cpp</a>      | ?? |
| <a href="#">Factory.h</a>        | ?? |
| <a href="#">General.cpp</a>      | ?? |
| <a href="#">General.h</a>        | ?? |
| <a href="#">KeyPoint.cpp</a>     | ?? |
| <a href="#">KeyPoint.h</a>       | ?? |
| <a href="#">Negotiator.cpp</a>   | ?? |
| <a href="#">Negotiator.h</a>     | ?? |
| <a href="#">NegotiatorTest.h</a> | ?? |
| <a href="#">Passive.cpp</a>      | ?? |

|                      |    |
|----------------------|----|
| Passive.h            | ?? |
| Personnel.cpp        | ?? |
| Personnel.h          | ?? |
| PersonnelFactory.cpp | ?? |
| PersonnelFactory.h   | ?? |
| Piercing.cpp         | ?? |
| Piercing.h           | ?? |
| Rainy.cpp            | ?? |
| Rainy.h              | ?? |
| RoundStats.cpp       | ?? |
| RoundStats.h         | ?? |
| SaveArchive.cpp      | ?? |
| SaveArchive.h        | ?? |
| Strategy.cpp         | ?? |
| Strategy.h           | ?? |
| Sunny.cpp            | ?? |
| Sunny.h              | ?? |
| Support.cpp          | ?? |
| Support.h            | ?? |
| SupportFactory.cpp   | ?? |
| SupportFactory.h     | ?? |
| TerrainType.cpp      | ?? |
| TerrainType.h        | ?? |
| testmain.cpp         | ?? |
| Type.cpp             | ?? |
| Type.h               | ?? |
| Vehicle.cpp          | ?? |
| Vehicle.h            | ?? |
| VehicleFactory.cpp   | ?? |
| VehicleFactory.h     | ?? |
| WarEngine.cpp        | ?? |
| WarEngine.h          | ?? |
| WarEngineMemento.cpp | ?? |
| WarEngineMemento.h   | ?? |
| WarEngineState.cpp   | ?? |
| WarEngineState.h     | ?? |
| WarTheatre.cpp       | ?? |
| WarTheatre.h         | ?? |
| Weather.cpp          | ?? |
| Weather.h            | ?? |

## Chapter 4

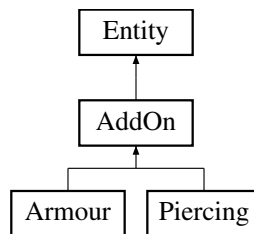
# Class Documentation

### 4.1 AddOn Class Reference

[AddOn](#) class.

```
#include <AddOn.h>
```

Inheritance diagram for AddOn:



#### Public Member Functions

- [AddOn](#) (int value)  
*Instantiates an [AddOn](#).*
- void [setValue](#) (int value)  
*Sets the [AddOn](#)'s value attribute.*
- int [getValue](#) ()  
*Returns the [AddOn](#)'s value attribute.*
- void [setEntity](#) ([Entity](#) \*entity)  
*Sets the [AddOn](#)'s entity attribute.*
- [Entity](#) \* [getEntity](#) ()  
*Returns the [AddOn](#)'s entity attribute.*
- virtual void [takeDamage](#) (int damage)=0
- virtual void [dealDamage](#) ([Entity](#) \*entity)=0
- virtual [AddOn](#) \* [clone](#) ()=0

#### Protected Attributes

- int [value](#)
- [Entity](#) \* [entity](#)

### 4.1.1 Detailed Description

[AddOn](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 10 of file [AddOn.h](#).

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 AddOn()

```
AddOn::AddOn (
    int value )
```

Instantiates an [AddOn](#).

##### Parameters

|              |                |
|--------------|----------------|
| <i>value</i> | must be an int |
|--------------|----------------|

Definition at line 3 of file [AddOn.cpp](#).

```
00003      : Entity() {
00004      this->value = value;
00005      entity = NULL;
00006 }
```

### 4.1.3 Member Function Documentation

#### 4.1.3.1 clone()

```
virtual AddOn * AddOn::clone ( ) [pure virtual]
```

Implements [Entity](#).

Implemented in [Armour](#), and [Piercing](#).

#### 4.1.3.2 dealDamage()

```
virtual void AddOn::dealDamage (
    Entity * entity ) [pure virtual]
```

Implements [Entity](#).

Implemented in [Armour](#), and [Piercing](#).

#### 4.1.3.3 getEntity()

```
Entity * AddOn::getEntity ( )
```

Returns the [AddOn](#)'s entity attribute.

Postconditions:

- Returns the entity attribute of the [AddOn](#) object

Returns

Entity\* The entity of the [AddOn](#)

Definition at line 20 of file [AddOn.cpp](#).

```
00020 {  
00021     return this->entity;  
00022 }
```

#### 4.1.3.4 getValue()

```
int AddOn::getValue ( )
```

Returns the [AddOn](#)'s value attribute.

Postconditions:

- Returns the value attribute of the [AddOn](#) object

Returns

int The values of the [AddOn](#)

Definition at line 12 of file [AddOn.cpp](#).

```
00012 {  
00013     return value;  
00014 }
```

#### 4.1.3.5 setEntity()

```
void AddOn::setEntity (  
    Entity * entity )
```

Sets the [AddOn](#)'s entity attribute.

Preconditions:

- entity must be an Entity\*

Postconditions:

- Sets the entity attribute of the [AddOn](#) object to the passed in entity

**Parameters**

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

**Returns**

void

Definition at line 16 of file [AddOn.cpp](#).

```
00016 {  
00017     this->entity = entity;  
00018 }
```

**4.1.3.6 setValue()**

```
void AddOn::setValue (  
    int value )
```

Sets the [AddOn](#)'s value attribute.

Preconditions:

- value must be an int

Postconditions:

- Sets the value attribute of the [AddOn](#) object to the passed in value

**Parameters**

|              |                |
|--------------|----------------|
| <i>value</i> | must be an int |
|--------------|----------------|

**Returns**

void

Definition at line 8 of file [AddOn.cpp](#).

```
00008 {  
00009     this->value = value;  
00010 }
```

**4.1.3.7 takeDamage()**

```
virtual void AddOn::takeDamage (  
    int damage ) [pure virtual]
```

Implements [Entity](#).Implemented in [Armour](#), and [Piercing](#).



### 4.1.4 Member Data Documentation

#### 4.1.4.1 entity

```
Entity* AddOn::entity [protected]
```

Definition at line 14 of file [AddOn.h](#).

#### 4.1.4.2 value

```
int AddOn::value [protected]
```

Definition at line 13 of file [AddOn.h](#).

The documentation for this class was generated from the following files:

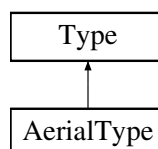
- [AddOn.h](#)
- [AddOn.cpp](#)

## 4.2 AerialType Class Reference

[AerialType](#) class.

```
#include <AerialType.h>
```

Inheritance diagram for [AerialType](#):



### Public Member Functions

- [AerialType](#) ()  
*Instantiates the ariel type.*
- string [getTypeDesc](#) ()  
*Returns ariel type description.*
- [Type](#) \* [clone](#) ()

### 4.2.1 Detailed Description

[AerialType](#) class.

Used to define [Entity](#) objects as ariel type.

Definition at line 11 of file [AerialType.h](#).

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 AerialType()

```
AerialType::AerialType ( )
```

Instantiates the ariel type.

Definition at line 3 of file [AerialType.cpp](#).

```
00003 {}
```

### 4.2.3 Member Function Documentation

#### 4.2.3.1 clone()

```
Type * AerialType::clone ( ) [virtual]
```

Implements [Type](#).

Definition at line 9 of file [AerialType.cpp](#).

```
00009 {  
00010     return new AerialType();  
00011 }
```

#### 4.2.3.2 getTypeDesc()

```
string AerialType::getTypeDesc ( ) [virtual]
```

Returns ariel type description.

Postconditions:

- Returns the ariel type

**Returns**

string The ariel type string

Implements [Type](#).

Definition at line 5 of file [AerialType.cpp](#).

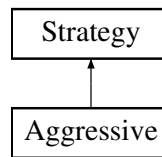
```
00005 {  
00006     return "Aerial";  
00007 }
```

The documentation for this class was generated from the following files:

- [AerialType.h](#)
- [AerialType.cpp](#)

## 4.3 Aggressive Class Reference

Inheritance diagram for Aggressive:



### Public Member Functions

- void `performStrat` (`KeyPoint` \*keyPoint, `Alliance` \*alliance)  
*This function will perform an [Aggressive](#) strategy.*
- `Strategy` \* `clone` ()  
*Returns the clone of the [Aggressive Strategy](#) object.*

### Additional Inherited Members

#### 4.3.1 Detailed Description

Definition at line 5 of file [Aggressive.h](#).

#### 4.3.2 Constructor & Destructor Documentation

##### 4.3.2.1 Aggressive()

```
Aggressive::Aggressive ( )
```

Definition at line 4 of file [Aggressive.cpp](#).  
00004 {}

#### 4.3.3 Member Function Documentation

#### 4.3.3.1 clone()

```
Strategy * Aggressive::clone ( ) [virtual]
```

Returns the clone of the [Aggressive Strategy](#) object.

##### Returns

Strategy\* The clone of the [Aggressive Strategy](#) object

Implements [Strategy](#).

Definition at line 11 of file [Aggressive.cpp](#).

```
00011 {
00012     return new Aggressive();
00013 }
```

#### 4.3.3.2 performStrat()

```
void Aggressive::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [virtual]
```

This function will perform an [Aggressive](#) strategy.

##### Preconditions:

- Takes in object of type [KeyPoint](#) as parameter

##### Postconditions:

- Returns the [Strategy](#) type

##### Parameters

|                 |   |
|-----------------|---|
| <i>keyPoint</i> | an <a href="#">Aggressive</a> strategy will then be performed at this specific keypoint |
|-----------------|---|

##### Returns

void The function will return a void

Implements [Strategy](#).

Definition at line 6 of file [Aggressive.cpp](#).

```
00006 {
00007     int randomNumber = (rand() % 10) + 5;
00008     keyPoint->moveEntitiesInto(alliance, randomNumber);
00009 }
```

The documentation for this class was generated from the following files:

- [Aggressive.h](#)
- [Aggressive.cpp](#)

## 4.4 Alliance Class Reference

### Public Member Functions

- [Alliance](#) ()  
*Instantiates the [Alliance](#).*
- [Alliance](#) ([Alliance](#) &alliance)  
*Instantiates a copy of an [Alliance](#).*
- [~Alliance](#) ()  
*Destructor for the [Alliance](#) object.*
- void [setNegotiator](#) ([Negotiator](#) \*newNegotiator)  
*Sets the entity negotiator.*
- void [addCountry](#) ([Country](#) \*nation)  
*Adds a country into the members vector which holds countries.*
- vector< [Entity](#) \* > [getReserveEntities](#) (int number)  
*Return a given number of reserve entites vector.*
- void [addReserveEntity](#) ([Entity](#) \*entity)  
*Adds a entity to the reserve entities.*
- int [numRemainingEntities](#) ()
- bool [considerPeace](#) ()  
*Considers to stop war with the allaince passed into the function header.*
- void [addFactory](#) ([Factory](#) \*factory)  
*Adds a factory into the production vector which holds factories.*
- void [runFactories](#) ()
- void [surrender](#) ()  
*Makes the current alliance give up of the war by surrendering.*
- int [getID](#) ()  
*Returns [Alliance](#)'s aID.*
- bool [offerPeace](#) ()  
*Offers peace to stop war with the alliance fighting against using [sendPeace](#).*
- [Alliance](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current [Alliance](#).*
- void [setActiveStatus](#) (bool active)  
*Sets variable active to the passed in parameter.*
- int [getActive](#) ()

### 4.4.1 Detailed Description

Definition at line 13 of file [Alliance.h](#).

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 Alliance() [1/2]

```
Alliance::Alliance ( )
```

Instantiates the [Alliance](#).

Definition at line 12 of file [Alliance.cpp](#).

```
00012     {
00013         this->active = 1;
00014         this->aID = totalNum++;
00015         this->negotiator = NULL;
00016         srand(time(0));
00017     }
```

#### 4.4.2.2 Alliance() [2/2]

```
Alliance::Alliance (
    Alliance & alliance )
```

Instantiates a copy of an [Alliance](#).

##### Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>alliance</i> | must be an alliance instance |
|-----------------|------------------------------|

Definition at line 19 of file [Alliance.cpp](#).

```
00019     {
00020         this->active = alliance.active;
00021         this->aID = alliance.aID;
00022
00023         for (int i = 0; i < alliance.members.size(); i++)
00024             this->addCountry(alliance.members[i]->clone());
00025
00026         for (int i = 0; i < alliance.production.size(); i++)
00027             this->addFactory(alliance.production[i]->clone());
00028
00029         for (int i = 0; i < alliance.reserveEntities.size(); i++)
00030             this->addReserveEntity(alliance.reserveEntities[i]->clone());
00031
00032         this->negotiator = NULL;
00033     }
```

#### 4.4.2.3 ~Alliance()

```
Alliance::~~Alliance ( )
```

Destructor for the [Alliance](#) object.

Definition at line 35 of file [Alliance.cpp](#).

```
00035     {
00036
00037         for (int i = 0; i < members.size(); i++)
00038             //delete members[i];
00039
00040         if (this->negotiator != NULL) {
00041             this->negotiator->removeAlliance(this);
00042
00043             if (this->negotiator->getNumAlliances() == 1)
00044                 delete this->negotiator;
00045         }
00046     }
```

### 4.4.3 Member Function Documentation

#### 4.4.3.1 addCountry()

```
void Alliance::addCountry (
    Country * nation )
```

Adds a country into the members vector which holds countries.

Preconditions:

- nation must be an Country\*

Postconditions:

- Country is added to the members vector

Parameters

|        |                     |
|--------|---------------------|
| nation | must be an Country* |
|--------|---------------------|

Returns

void

Definition at line 52 of file Alliance.cpp.

```
00052                                     {
00053     members.push_back(nation);
00054 }
```

#### 4.4.3.2 addFactory()

```
void Alliance::addFactory (
    Factory * factory )
```

Adds a factory into the production vector which holds factories.

Preconditions:

- f must be an Factory\*

Postconditions:

- Factory is added to the production vector

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>factory</i> | must be a Factory* |
|----------------|--------------------|

**Returns**

void

Definition at line 78 of file [Alliance.cpp](#).

```
00078                                     {  
00079     production.push_back(factory);  
00080 }
```

**4.4.3.3 addReserveEntity()**

```
void Alliance::addReserveEntity (  
    Entity * entity )
```

Adds a entity to the reserve entities.

Preconditions:

- nation must be an Entity\*

Postconditions:

- [Entity](#) is added to the reserveEntities vector

**Parameters**

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

**Returns**

void

Definition at line 66 of file [Alliance.cpp](#).

```
00066                                     {  
00067     reserveEntities.push_back(entity);  
00068 }
```

**4.4.3.4 clone()**

```
Alliance * Alliance::clone ( )
```

Instantiates and returns a clone of the current [Alliance](#).

Postconditions:



- Returns the clone of the current [Alliance](#)

#### Returns

Alliance\* The alliance clone

Definition at line 114 of file [Alliance.cpp](#).

```
00114         {  
00115     return new Alliance(*this);  
00116 }
```

#### 4.4.3.5 considerPeace()

```
bool Alliance::considerPeace ( )
```

Considers to stop war with the alliance passed into the function header.

Preconditions:

- id must be an integer

Postconditions:

- Result of consideration returned in the form of a bool

#### Returns

bool

Definition at line 74 of file [Alliance.cpp](#).

```
00074     {  
00075     return (rand() % 2 == 0);  
00076 }
```

#### 4.4.3.6 getActive()

```
int Alliance::getActive ( )
```

Definition at line 110 of file [Alliance.cpp](#).

```
00110     {  
00111     return active;  
00112 }
```

#### 4.4.3.7 getID()

```
int Alliance::getID ( )
```

Returns [Alliance](#)'s aID.

Postconditions:

- Returns the aID

##### Returns

int The ID of the [Alliance](#) object

Definition at line 95 of file [Alliance.cpp](#).

```
00095     {
00096     return this->aID;
00097 }
```

#### 4.4.3.8 getReserveEntities()

```
vector< Entity * > Alliance::getReserveEntities (
    int number )
```

Return a given number of reserve entites vector.

Precondition:

- number must be an int

Postconditions:

- Return a given number of reserve entities
- If not enough reseerves return amount available

##### Parameters

|               |                |
|---------------|----------------|
| <i>number</i> | must be an int |
|---------------|----------------|

##### Returns

vector<Entity\*>\*

Definition at line 56 of file [Alliance.cpp](#).

```
00056     {
00057     vector<Entity*> out;
00058     for (int i = 0; i < number && i < reserveEntities.size(); i++) {
00059         out.push_back(reserveEntities[i]);
```

```

00060         reserveEntities.erase(reserveEntities.begin() + i);
00061     }
00062
00063     return out;
00064 }

```

#### 4.4.3.9 numRemainingEntities()

```
int Alliance::numRemainingEntities ( )
```

Definition at line 70 of file [Alliance.cpp](#).

```

00070     {
00071         return reserveEntities.size();
00072     }

```

#### 4.4.3.10 offerPeace()

```
bool Alliance::offerPeace ( )
```

Offers peace to stop war with the alliance fighting against using `sendPeace`.

Postconditions:

- Result of consideration returned from the enemy alliance which considered peace

#### Returns

bool

Definition at line 99 of file [Alliance.cpp](#).

```

00099     {
00100
00101         if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
00102         {
00103             this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
00104             return true;
00105         }
00106
00107         return false;
00108     }

```

#### 4.4.3.11 runFactories()

```
void Alliance::runFactories ( )
```

Definition at line 82 of file [Alliance.cpp](#).

```

00082     {
00083         for (int i = 0; i < production.size(); i++) {
00084             RoundStats::numEntitiesCreated++;
00085             reserveEntities.push_back(production[i]->createEntity(this));
00086         }
00087     }

```

#### 4.4.3.12 setActiveStatus()

```
void Alliance::setActiveStatus (
    bool active )
```

Sets variable active to the passed in parameter.

PreConditions:

- active must be an a bool

PostConditions:

- The variable active is set to the passed in the parameter

Parameters

|           |                  |
|-----------|------------------|
| <i>ID</i> | a bool parameter |
|-----------|------------------|

#### 4.4.3.13 setNegotiator()

```
void Alliance::setNegotiator (
    Negotiator * newNegotiator )
```

Sets the entity negotiator.

Preconditions:

- n must be an Negotiator\*

Postconditions:

- Sets the negotiator of the [Alliance](#) object

Parameters

|          |                       |
|----------|-----------------------|
| <i>n</i> | must be a Negotiator* |
|----------|-----------------------|

Returns

void

Definition at line 48 of file [Alliance.cpp](#).

```
00048
00049     this->negotiator = negotiator;
00050 }
```

#### 4.4.3.14 surrender()

```
void Alliance::surrender ( )
```

Makes the current alliance give up of the war by surrendering.

Postconditions:

- Sets the active variable to false
- Removes this alliance from the [Negotiator](#) vector

#### Returns

void

Definition at line 89 of file [Alliance.cpp](#).

```
00089     {
00090         this->active = 2; //Number 2 means that Alliance has surrendered
00091
00092         this->negotiator->removeAlliance(this);
00093     }
```

The documentation for this class was generated from the following files:

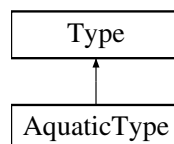
- Alliance.h
- Alliance.cpp

## 4.5 AquaticType Class Reference

[AquaticType](#) class.

```
#include <AquaticType.h>
```

Inheritance diagram for [AquaticType](#):



### Public Member Functions

- [AquaticType](#) ()  
*Instantiates the aquatic type.*
- string [getTypeDesc](#) ()  
*Returns aquatic type description.*
- [Type](#) \* [clone](#) ()

### 4.5.1 Detailed Description

[AquaticType](#) class.

Used to define [Entity](#) objects as aquatic type.

Definition at line 11 of file [AquaticType.h](#).

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 AquaticType()

```
AquaticType::AquaticType ( )
```

Instantiates the aquatic type.

Definition at line 5 of file [AquaticType.cpp](#).

```
00005 {}
```

### 4.5.3 Member Function Documentation

#### 4.5.3.1 clone()

```
Type * AquaticType::clone ( ) [virtual]
```

Implements [Type](#).

Definition at line 11 of file [AquaticType.cpp](#).

```
00011 {  
00012     return new AquaticType();  
00013 }
```

#### 4.5.3.2 getTypeDesc()

```
string AquaticType::getTypeDesc ( ) [virtual]
```

Returns aquatic type description.

Postconditions:

- Returns the aquatic type

Returns

string The aquatic type string

Implements [Type](#).

Definition at line 7 of file [AquaticType.cpp](#).

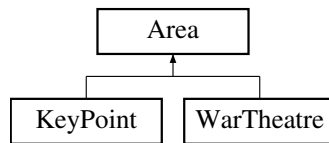
```
00007 {  
00008     return "Aquatic";  
00009 }
```

The documentation for this class was generated from the following files:

- [AquaticType.h](#)
- [AquaticType.cpp](#)

## 4.6 Area Class Reference

Inheritance diagram for Area:



### Public Member Functions

- [Area](#) (std::string areaName)  
*Instantiates the area.*
- virtual [~Area](#) ()  
*Destroys the area object.*
- virtual bool [isKeyPoint](#) ()=0
- virtual void [simulateBattle](#) ([Alliance](#) \*alliance)=0
- std::string [getAreaName](#) () const  
*Get the [Area Type](#) object.*
- virtual [Area](#) \* [clone](#) ()=0
- virtual void [addGeneral](#) ([General](#) \*general)=0

### 4.6.1 Detailed Description

Definition at line 8 of file [Area.h](#).

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 Area()

```
Area::Area (  
    std::string areaName )
```

Instantiates the area.

Definition at line 5 of file [Area.cpp](#).

```
00005     {  
00006     this->areaName = areaName;  
00007 }
```

#### 4.6.2.2 ~Area()

```
Area::~~Area ( ) [virtual]
```

Destroys the area object.

Definition at line 9 of file [Area.cpp](#).

```
00009 {}
```

### 4.6.3 Member Function Documentation

#### 4.6.3.1 addGeneral()

```
virtual void Area::addGeneral (
    General * general ) [pure virtual]
```

Implemented in [WarTheatre](#).

#### 4.6.3.2 clone()

```
virtual Area * Area::clone ( ) [pure virtual]
```

Implemented in [KeyPoint](#), and [WarTheatre](#).

#### 4.6.3.3 getAreaName()

```
std::string Area::getAreaName ( ) const
```

Get the [Area Type](#) object.

##### Returns

std::string reaturns the type

Definition at line 11 of file [Area.cpp](#).

```
00011 {
00012     return areaName;
00013 }
```



#### 4.6.3.4 isKeyPoint()

```
virtual bool Area::isKeyPoint ( ) [pure virtual]
```

Implemented in [KeyPoint](#), and [WarTheatre](#).

#### 4.6.3.5 simulateBattle()

```
virtual void Area::simulateBattle (
    Alliance * alliance ) [pure virtual]
```

Implemented in [KeyPoint](#), and [WarTheatre](#).

The documentation for this class was generated from the following files:

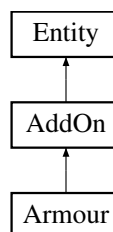
- Area.h
- Area.cpp

## 4.7 Armour Class Reference

[Armour](#) class.

```
#include <Armour.h>
```

Inheritance diagram for [Armour](#):



### Public Member Functions

- [Armour](#) (int value)  
*Instantiates an [Armour](#).*
- void [takeDamage](#) (int damage)  
*Decreases the entities' armour value (or health when their armour has depleted)*
- void [dealDamage](#) ([Entity](#) \*entity)  
*Adds to the damage [Entity](#) objects inflict.*
- [AddOn](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current [Armour](#).*

## Additional Inherited Members

### 4.7.1 Detailed Description

[Armour](#) class.

Used to add protective armour to [Entity](#) objects.

Definition at line 11 of file [Armour.h](#).

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 Armour()

```
Armour::Armour (
    int value )
```

Instantiates an [Armour](#).

##### Parameters

|              |                |
|--------------|----------------|
| <i>value</i> | must be an int |
|--------------|----------------|

Definition at line 3 of file [Armour.cpp](#).

```
00003 :   AddOn(value) {}
```

### 4.7.3 Member Function Documentation

#### 4.7.3.1 clone()

```
AddOn * Armour::clone ( ) [virtual]
```

Instantiates and returns a clone of the current [Armour](#).

Postconditions:

- Returns the clone of the current [Armour](#)

##### Returns

[Armour](#)\* The [Armour](#) clone

Implements [AddOn](#).

Definition at line 17 of file [Armour.cpp](#).

```
00017 {
00018     Armour* armour = new Armour(value);
00019     if (getEntity() != NULL)
00020         armour->setEntity(entity->clone());
00021     return armour;
00022 }
```

#### 4.7.3.2 dealDamage()

```
void Armour::dealDamage (
    Entity * entity ) [virtual]
```

Adds to the damage [Entity](#) objects inflict.

Preconditions:

- entity must be an Entity\*

Postconditions:

- Does nothing

Parameters

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

Returns

void

Implements [AddOn](#).

Definition at line 13 of file [Armour.cpp](#).

```
00013 {
00014     this->entity->dealDamage(entity);
00015 }
```

#### 4.7.3.3 takeDamage()

```
void Armour::takeDamage (
    int damage ) [virtual]
```

Decreases the entities' armour value (or health when their armour has depleted)

Preconditions:

- damage must be an int

Postconditions:

- Decreases the entities' armour value (or health when their armour has diminished) by the passed in value

**Parameters**

|               |                |
|---------------|----------------|
| <i>damage</i> | must be an int |
|---------------|----------------|

**Returns**

void

Implements [AddOn](#).Definition at line 5 of file [Armour.cpp](#).

```

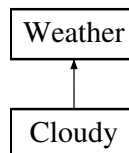
00005
00006     if (value > 0) {
00007         value -= damage;
00008     } else {
00009         entity->takeDamage (damage);
00010     }
00011 }
```

The documentation for this class was generated from the following files:

- [Armour.h](#)
- [Armour.cpp](#)

## 4.8 Cloudy Class Reference

Inheritance diagram for Cloudy:

**Public Member Functions**

- [Cloudy](#) ()  
*Instantiates the [Cloudy](#) object of the state pattern.*
- `std::string getWeather ()`  
*Returns string which tells us the weather.*
- `void handleChange (KeyPoint *k)`  
*Will change the current state of the weather inside the specific keypoint.*
- `Weather * clone ()`  
*Returns a clone of the [Cloudy](#) object.*

**Additional Inherited Members**

### 4.8.1 Detailed Description

Definition at line 6 of file [Cloudy.h](#).

## 4.8.2 Constructor & Destructor Documentation

### 4.8.2.1 Cloudy()

```
Cloudy::Cloudy ( )
```

Instantiates the [Cloudy](#) object of the state pattern.

Definition at line 4 of file [Cloudy.cpp](#).

```
00004         : Weather() {  
00005     this->multiplier = 0.75;  
00006 }
```

## 4.8.3 Member Function Documentation

### 4.8.3.1 clone()

```
Weather * Cloudy::clone ( ) [virtual]
```

Returns a clone of the [Cloudy](#) object.

#### Returns

Weather\* Clone of cloudy object

Implements [Weather](#).

Definition at line 17 of file [Cloudy.cpp](#).

```
00017     {  
00018     return new Cloudy();  
00019 }
```

### 4.8.3.2 getWeather()

```
std::string Cloudy::getWeather ( ) [virtual]
```

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

#### Returns

std::string which is the current state

Implements [Weather](#).

Definition at line 8 of file [Cloudy.cpp](#).

```
00008     {  
00009     return "Cloudy";  
00010 }
```

#### 4.8.3.3 handleChange()

```
void Cloudy::handleChange (
    KeyPoint * k ) [virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- k must be a KeyPoint\*

Postconditions:

- Changes the current weather to the next one in the state pattern ([Rainy](#))

##### Parameters

|          |                     |
|----------|---------------------|
| <i>k</i> | must be a KeyPoint* |
|----------|---------------------|

##### Returns

void

Implements [Weather](#).

Definition at line 12 of file [Cloudy.cpp](#).

```
00012
00013     Rainy* newWeather = new Rainy();
00014     k->setWeather(newWeather);
00015 }
```

The documentation for this class was generated from the following files:

- Cloudy.h
- Cloudy.cpp

## 4.9 Country Class Reference

### Public Member Functions

- [Country](#) (std::string name)  
*Instantiates the [Country](#).*
- [Country](#) \* clone ()  
*Instantiates and returns a clone of the current [Country](#).*
- std::string [getName](#) () const  
*Get the name of the country.*
- int [getID](#) () const  
*Get the id of the country.*

### 4.9.1 Detailed Description

Definition at line 5 of file [Country.h](#).

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 Country()

```
Country::Country (
    std::string name )
```

Instantiates the [Country](#).

##### Parameters

|             |                  |
|-------------|------------------|
| <i>name</i> | must be a string |
|-------------|------------------|

Definition at line 5 of file [Country.cpp](#).

```
00005 {
00006     this->name = name;
00007     this->id = rand() % 1000;
00008 }
```

### 4.9.3 Member Function Documentation

#### 4.9.3.1 clone()

```
Country * Country::clone ( )
```

Instantiates and returns a clone of the current [Country](#).

Postconditions:

- Returns the clone of the current [Country](#)

##### Returns

[Country](#)\* The country clone

Definition at line 11 of file [Country.cpp](#).

```
00011 {
00012     return new Country(this->name);
00013 }
```

#### 4.9.3.2 getID()

```
int Country::getID ( ) const
```

Get the id of the country.

PostConditions:

- return the id the id of the country

##### Returns

int

Definition at line 19 of file [Country.cpp](#).

```
00019 {  
00020     return this->id;  
00021 }
```

#### 4.9.3.3 getName()

```
string Country::getName ( ) const
```

Get the name of the country.

PostConditions:

- Return the name of the country

##### Returns

string

Definition at line 15 of file [Country.cpp](#).

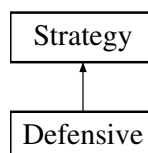
```
00015 {  
00016     return this->name;  
00017 }
```

The documentation for this class was generated from the following files:

- Country.h
- Country.cpp

## 4.10 Defensive Class Reference

Inheritance diagram for Defensive:





## Public Member Functions

- void [performStrat](#) ([KeyPoint](#) \*keyPoint, [Alliance](#) \*alliance)  
This function will perform an [Defensive](#) strategy.
- [Strategy](#) \* [clone](#) ()  
Returns the clone of the *Defensive* [Strategy](#) object.

## Additional Inherited Members

### 4.10.1 Detailed Description

Definition at line 7 of file [Defensive.h](#).

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 Defensive()

```
Defensive::Defensive ( )
```

Definition at line 3 of file [Defensive.cpp](#).

```
00003         {
00004
00005     }
```

### 4.10.3 Member Function Documentation

#### 4.10.3.1 clone()

```
Strategy * Defensive::clone ( ) [virtual]
```

Returns the clone of the *Defensive* [Strategy](#) object.

#### Returns

[Strategy](#)\* The clone of the [Defensive Strategy](#) object

Implements [Strategy](#).

Definition at line 13 of file [Defensive.cpp](#).

```
00013         {
00014     return new Defensive();
00015 }
```

#### 4.10.3.2 performStrat()

```
void Defensive::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [virtual]
```

This function will perform an [Defensive](#) strategy.

#### Author

Antwi-Antwi

**Parameters**

|                 |  |
|-----------------|--|
| <i>keyPoint</i> | an <a href="#">Defensive</a> strategy will then be performed at this specific keypoint |
|-----------------|--|

**Returns**

void The function will return a void

Implements [Strategy](#).

Definition at line 7 of file [Defensive.cpp](#).

```
00007                                     {
00008
00009     int randomNumber = (rand() % 5) + 1;
00010     keyPoint->moveEntitiesInto(alliance, randomNumber);
00011 }
```

The documentation for this class was generated from the following files:

- [Defensive.h](#)
- [Defensive.cpp](#)

## 4.11 EasySetup Class Reference

### Public Member Functions

- void [setupSimulation](#) ()
- void [runSimulation](#) ()
- void [loadPrevSave](#) ()
- void [loadSpecificSave](#) (std::string name)
- void [saveSimulationSetup](#) ()

#### 4.11.1 Detailed Description

Definition at line 11 of file [EasySetup.h](#).

#### 4.11.2 Constructor & Destructor Documentation

##### 4.11.2.1 EasySetup()

`EasySetup::EasySetup ( )`

Definition at line 21 of file [EasySetup.cpp](#).

```
00021     {
00022         saveArchive = new SaveArchive();
00023     }
```

### 4.11.3 Member Function Documentation

#### 4.11.3.1 loadPrevSave()

```
void EasySetup::loadPrevSave ( )
```

Definition at line 284 of file [EasySetup.cpp](#).

```
00284         {
00285
00286     try{
00287         WarEngineMemento* saveFile = saveArchive->getLastSave();
00288         WarEngine::getInstance().loadSave(saveFile);
00289     }
00290     catch(const std::exception& error){
00291         std::cerr << error.what() << "\n";
00292     }
00293 }
00294
00295
00296 }
```

#### 4.11.3.2 loadSpecificSave()

```
void EasySetup::loadSpecificSave (
    std::string name )
```

Definition at line 298 of file [EasySetup.cpp](#).

```
00298         {
00299
00300     try{
00301         WarEngineMemento* saveFile = saveArchive->getSave(name);
00302         WarEngine::getInstance().loadSave(saveFile);
00303     }
00304     catch(const std::out_of_range& range_error){
00305         std::cerr << range_error.what() << "\n";
00306     }
00307 }
00308
00309
00310
00311 }
```

#### 4.11.3.3 runSimulation()

```
void EasySetup::runSimulation ( )
```

Definition at line 267 of file [EasySetup.cpp](#).

```
00267         {
00268
00269     WarEngine::getInstance().simulate();
00270 }
```

#### 4.11.3.4 saveSimulationSetup()

```
void EasySetup::saveSimulationSetup ( )
```

Definition at line 272 of file [EasySetup.cpp](#).

```
00272     {
00273
00274         // Getting the name of the save
00275         cout << "Please enter name of save: ";
00276         string saveName;
00277         getline(cin, saveName);
00278
00279         // saving the current state of the simulation
00280         saveArchive->addNewSave(saveName, WarEngine::getInstance().saveState());
00281
00282     }
```

#### 4.11.3.5 setupSimulation()

```
void EasySetup::setupSimulation ( )
```

Definition at line 25 of file [EasySetup.cpp](#).

```
00025     {
00026         while (true)
00027         {
00028             cout << "Load simulation (L) or New Simulation (N): ";
00029             string selectedOption;
00030             cin >> selectedOption;
00031             cin.ignore();
00032
00033             if(toupper(selectedOption[0]) == 'L')
00034             {
00035                 string saveName;
00036                 cout << "Please enter the name of the save to be re-simulated" << endl;
00037                 getline(cin, saveName); // getting the name of the save-archive
00038                 try {
00039                     this->loadSpecificSave(saveName); // loading the save-archive
00040                     return; // will return if the above the function does not throw an exception
00041                 } catch(const std::exception& exception) {
00042                     cout << "Error: " << exception.what() << endl;
00043
00044                     if (strcmp(exception.what(), "Save archive is empty") == 0) {
00045                         cout << "Please create new simulation" << endl;
00046                         goto setup;
00047                     } else if (strcmp(exception.what(), "No save with given name exists") == 0) {
00048
00049                         cout << "Please enter the correct name of save-archive and try again or create new
00050 simulation" << endl;
00051                     }
00052                 }
00053
00054                 else if(toupper(selectedOption[0]) == 'N') {
00055                     // setting up a new simulation
00056                     goto setup;
00057                 } else {
00058                     cout << "Incorrect input: Please enter (L) or (N)" << endl;
00059                 }
00060             }
00061
00062             setup:
00063             // Creating alliances and generals
00064             int numAlliesAndGenerals;
00065             cout << "Enter number of alliances: ";
00066             cin >> numAlliesAndGenerals;
00067
00068             Alliance** alliances = new Alliance*[numAlliesAndGenerals];
00069             General** generals = new General*[numAlliesAndGenerals];
00070
00071             int numCountries,
00072                 numFactories;
00073             string countryName,
00074                 factoryType,
00075                 selectedFactory,
00076                 selectedAddOn;
00077             Country* country;
```

```

00078     Type* type;
00079     AddOn* addOn;
00080     Factory* factory;
00081
00082     Negotiator* negotiator = new Negotiator();
00083
00084     for (int i = 0; i < numAlliesAndGenerals; i++) {
00085         alliances[i] = new Alliance();
00086         negotiator->addAlliance(alliances[i]);
00087         alliances[i]->setNegotiator(negotiator);
00088         WarEngine::getInstance().addAlliance(alliances[i]);
00089
00090         cout << "Enter number of countries for Alliance " << alliances[i]->getID() << ": ";
00091         cin >> numCountries;
00092         cin.ignore();
00093
00094         for (int k = 0; k < numCountries; k++) {
00095             cout << "Enter name of county " << k+1 << ": ";
00096             getline(cin, countryName);
00097             country = new Country(countryName);
00098             alliances[i]->addCountry(country);
00099         }
00100
00101         cout << "Enter number of factories for Alliance " << alliances[i]->getID() << ": ";
00102         cin >> numFactories;
00103
00104         for (int k = 0; k < numFactories; k++) {
00105             retryType:
00106             cout << "Factory " << k+1 << " is of type Aquatic(Q), Aerial(E), or Terrain(T) : ";
00107             cin >> factoryType;
00108             cin.ignore();
00109
00110             if (toupper(factoryType[0]) == 'Q') {
00111                 type = new AerialType;
00112             } else if (toupper(factoryType[0]) == 'E') {
00113                 type = new AerialType;
00114             } else if (toupper(factoryType[0]) == 'T') {
00115                 type = new TerrainType;
00116             } else {
00117                 cout << "Invalid type input! Try again" << endl;
00118                 goto retryType;
00119             }
00120
00121             retryAddOn:
00122             cout << "Select AddOn for factory " << k+1 << " Armour(A), Piercing(P) or None(N) : ";
00123             getline(cin, selectedAddOn);
00124             if (toupper(selectedAddOn[0]) == 'A') {
00125                 int value;
00126                 cout << "Enter armour value: ";
00127                 cin >> value;
00128                 cin.ignore();
00129                 addOn = new Armour(value);
00130             } else if (toupper(selectedAddOn[0]) == 'P') {
00131                 int value;
00132                 cout << "Enter piercing value: ";
00133                 cin >> value;
00134                 cin.ignore();
00135                 addOn = new Piercing(value);
00136             } else if (toupper(selectedAddOn[0]) == 'N') {
00137                 addOn = NULL;
00138             } else {
00139                 cout << "Invalid AddOn input! Try again" << endl;
00140                 goto retryAddOn;
00141             }
00142
00143             retryFactory:
00144             cout << "Which factory is factory " << k+1 << " Vehicle(V), Personnel(P), or Support(S) : ";
00145             getline(cin, selectedFactory);
00146             if (toupper(selectedFactory[0]) == 'V') {
00147                 factory = new VehicleFactory(type, addOn);
00148             } else if (toupper(selectedFactory[0]) == 'P') {
00149                 factory = new PersonnelFactory(type, addOn);
00150             } else if (toupper(selectedFactory[0]) == 'S') {
00151                 factory = new SupportFactory(type, addOn);
00152             } else {
00153                 cout << "Invalid factory input! Try again" << endl;
00154                 goto retryFactory;
00155             }
00156
00157             alliances[i]->addFactory(factory);
00158         }
00159
00160         string selectedStrat;
00161         Strategy* strat;
00162
00163         retryStrat:

```

```

00164         cout << "What is this Alliances generals strategy Passive(P), Defensive(D), or
Aggressive(A) : ";
00165         getline(cin, selectedStrat);
00166         if (toupper(selectedStrat[0]) == 'P') {
00167             strat = new Passive();
00168         } else if (toupper(selectedStrat[0]) == 'D') {
00169             strat = new Defensive();
00170         } else if (toupper(selectedStrat[0]) == 'A') {
00171             strat = new Aggressive();
00172         } else {
00173             cout << "Invalid strategy input! Try again" << endl;
00174             goto retryStrat;
00175         }
00176
00177         generals[i] = new General(alliances[i], strat);
00178     }
00179
00180     int factoryRun;
00181     cout << "How many production runs do you wish to perform: ";
00182     cin >> factoryRun;
00183     cin.ignore();
00184     for (int i = 0; i < numAlliesAndGenerals; i++) {
00185         for (int j = 0; j < factoryRun; j++) {
00186             alliances[i]->runFactories();
00187         }
00188     }
00189
00190     // Creating main WarTheatre
00191     WarTheatre* mainBattleGround;
00192     cout << "Creating the main battle ground" << endl;
00193     string battleGroundName;
00194     cout << "Set main battle ground's name: ";
00195     getline(cin, battleGroundName);
00196     mainBattleGround = new WarTheatre(battleGroundName);
00197
00198     int sizeOfGrounds;
00199     cout << "Enter number of battle grounds in " << battleGroundName << " battle ground: ";
00200     cin >> sizeOfGrounds;
00201     cin.ignore();
00202     WarTheatre** battleGrounds = new WarTheatre*[sizeOfGrounds];
00203
00204     // Creating sub WarTheatres
00205     for (int i = 0; i < sizeOfGrounds; i++) {
00206         battleGroundName.clear();
00207         cout << "Set battle ground " << i+1 << "'s name: ";
00208         getline(cin, battleGroundName);
00209         battleGrounds[i] = new WarTheatre(battleGroundName);
00210     }
00211
00212     vector<int> numKeyPoints;
00213     int numKeyPoint = 0;
00214
00215     for (int i = 0; i < sizeOfGrounds; i++) {
00216         cout << "Enter number of key points in " << battleGrounds[i]->getAreaName() << " battle
ground: ";
00217         cin >> numKeyPoint;
00218         cin.ignore();
00219         numKeyPoints.push_back(numKeyPoint);
00220         numKeyPoint = 0;
00221     }
00222
00223     KeyPoint* keyPoint;
00224     string keyPointName;
00225     int numEntitiesInKeyPt;
00226
00227     // Creating KeyPoints for the sub WarTheatres
00228     for (int i = 0; i < sizeOfGrounds; i++) {
00229         numKeyPoint = numKeyPoints[i];
00230         cout << "For " << battleGrounds[i]->getAreaName() << "'s key points" << endl;
00231
00232         for (int k = 0; k < numKeyPoint; k++) {
00233             cout << "Set key point " << i+1 << "'s name: ";
00234             getline(cin, keyPointName);
00235             keyPoint = new KeyPoint(keyPointName);
00236
00237             for (int a = 0; a < numAlliesAndGenerals; a++) {
00238                 tryAgain:
00239                 cout << "There are " << alliances[a]->numRemainingEntities() << " for Alliance " <<
a+1 << endl;
00240                 cout << "How many would you like to place in " << keyPointName << " keypoint? ";
00241                 cin >> numEntitiesInKeyPt;
00242                 cin.ignore();
00243
00244                 if (alliances[a]->numRemainingEntities() > 0 &&
alliances[a]->numRemainingEntities() < numEntitiesInKeyPt) {
00245                     cout << "You selected more than the available amount. Try again " << endl;
00246                     goto tryAgain;

```

```

00247         } else if (alliances[a]->numRemainingEntities() <= 0) {
00248             continue;
00249         } else {
00250             keyPoint->moveEntitiesInto(alliances[a], numEntitiesInKeyPt);
00251         }
00252     }
00253     battleGrounds[i]->addArea(keyPoint);
00254 }
00255 }
00256 }
00257 mainBattleGround->addArea(battleGrounds[i]);
00258 }
00259 }
00260 for (int i = 0; i < numAlliesAndGenerals; i++) {
00261     mainBattleGround->addGeneral(generals[i]);
00262 }
00263 }
00264 WarEngine::getInstance().setWarTheatre(mainBattleGround);
00265 }

```

The documentation for this class was generated from the following files:

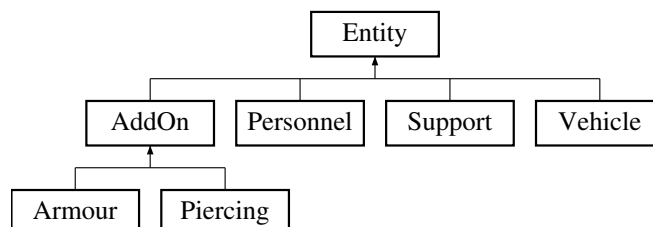
- EasySetup.h
- EasySetup.cpp

## 4.12 Entity Class Reference

[Entity](#) class.

```
#include <Entity.h>
```

Inheritance diagram for Entity:



### Public Member Functions

- [Entity](#) ()  
*Instantiates the entity.*
- [Entity](#) (Type \*type, int health, int damage)  
*Instantiates the entity.*
- Type \* [getType](#) ()  
*Returns entities type state.*
- void [setType](#) (Type \*type)  
*Sets the entities type state.*
- Alliance \* [getAlliance](#) ()  
*Returns entities alliance.*
- void [setAlliance](#) (Alliance \*alliance)  
*Sets the entities alliance.*
- int [getHealth](#) ()  
*Returns entities health.*

- void [setHealth](#) (int health)  
*Sets the entities health.*
- int [getDamage](#) ()  
*Returns entities damage.*
- void [setDamage](#) (int damage)  
*Sets the entities damage.*
- virtual void [takeDamage](#) (int damage)=0
- virtual void [dealDamage](#) ([Entity](#) \*entity)=0
- virtual [Entity](#) \* [clone](#) ()=0

### 4.12.1 Detailed Description

[Entity](#) class.

Used to simulate war entity objects.

Definition at line 13 of file [Entity.h](#).

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 [Entity\(\)](#) [1/2]

```
Entity::Entity ( )
```

Instantiates the entity.

Definition at line 5 of file [Entity.cpp](#).

```
00005      {
00006      health = 0;
00007      damage = 0;
00008      type = NULL;
00009  }
```

#### 4.12.2.2 [Entity\(\)](#) [2/2]

```
Entity::Entity (
    Type * type,
    int health,
    int damage )
```

Instantiates the entity.

#### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>type</i> | must be a <a href="#">Type</a> * |
|-------------|----------------------------------|



Definition at line 11 of file [Entity.cpp](#).

```
00011 {
00012     this->health = health;
00013     this->damage = damage;
00014     this->type = type;
00015 }
```

## 4.12.3 Member Function Documentation

### 4.12.3.1 clone()

```
virtual Entity * Entity::clone ( ) [pure virtual]
```

Implemented in [Armour](#), [Personnel](#), [Piercing](#), [Support](#), and [Vehicle](#).

### 4.12.3.2 dealDamage()

```
virtual void Entity::dealDamage (
    Entity * entity ) [pure virtual]
```

Implemented in [Armour](#), [Personnel](#), [Piercing](#), [Support](#), and [Vehicle](#).

### 4.12.3.3 getAlliance()

```
Alliance * Entity::getAlliance ( )
```

Returns entities alliance.

Postconditions:

- Returns the alliance

#### Returns

Type\* The alliance of the entity object

Definition at line 25 of file [Entity.cpp](#).

```
00025 {
00026     return this->alliance;
00027 }
```

#### 4.12.3.4 getDamage()

```
int Entity::getDamage ( )
```

Returns entities damage.

Postconditions:

- Returns the damage

##### Returns

int The damage of the entity object

Definition at line 41 of file [Entity.cpp](#).

```
00041     {  
00042     return this->damage;  
00043 }
```

#### 4.12.3.5 getHealth()

```
int Entity::getHealth ( )
```

Returns entities health.

Postconditions:

- Returns the health

##### Returns

int The health of the entity object

Definition at line 33 of file [Entity.cpp](#).

```
00033     {  
00034     return this->health;  
00035 }
```

#### 4.12.3.6 getType()

```
Type * Entity::getType ( )
```

Returns entities type state.

Postconditions:

- Returns the type

##### Returns

Type\* The type state of the entity object

Definition at line 17 of file [Entity.cpp](#).

```
00017     {  
00018     return this->type;  
00019 }
```

#### 4.12.3.7 setAlliance()

```
void Entity::setAlliance (
    Alliance * alliance )
```

Sets the entities alliance.

Preconditions:

- alliance must be an Alliance\*

Postconditions:

- Sets the alliance of the entity object

##### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>alliance</i> | must be a Alliance* |
|-----------------|---------------------|

##### Returns

void

Definition at line 29 of file [Entity.cpp](#).

```
00029 {
00030     this->alliance = alliance;
00031 }
```

#### 4.12.3.8 setDamage()

```
void Entity::setDamage (
    int damage )
```

Sets the entities damage.

Preconditions:

- damage must be an int

Postconditions:

- Sets the damage of the entity object

##### Parameters

|               |                |
|---------------|----------------|
| <i>damage</i> | must be an int |
|---------------|----------------|

**Returns**

void

Definition at line 45 of file [Entity.cpp](#).

```
00045      {  
00046          this->damage = damage;  
00047      }
```

**4.12.3.9 setHealth()**

```
void Entity::setHealth (  
    int health )
```

Sets the entities health.

Preconditions:

- health must be an int

Postconditions:

- Sets the health of the entity object

**Parameters**

|               |                |
|---------------|----------------|
| <i>health</i> | must be an int |
|---------------|----------------|

**Returns**

void

Definition at line 37 of file [Entity.cpp](#).

```
00037      {  
00038          this->health = health;  
00039      }
```

**4.12.3.10 setType()**

```
void Entity::setType (  
    Type * type )
```

Sets the entities type state.

Preconditions:

- type must be an Type\*

Postconditions:

- Sets the type state of the entity object

## Parameters

|             |                 |
|-------------|-----------------|
| <i>type</i> | must be a Type* |
|-------------|-----------------|

## Returns

void

Definition at line 21 of file [Entity.cpp](#).

```

00021     {
00022         this->type = type;
00023     }

```

## 4.12.3.11 takeDamage()

```

virtual void Entity::takeDamage (
    int damage ) [pure virtual]

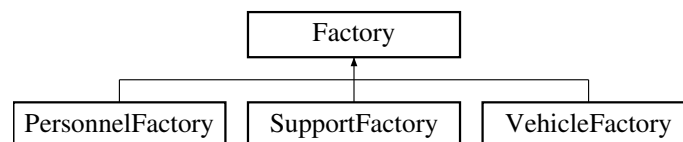
```

Implemented in [Armour](#), [Personnel](#), [Piercing](#), [Support](#), and [Vehicle](#).

The documentation for this class was generated from the following files:

- [Entity.h](#)
- [Entity.cpp](#)

## 4.13 Factory Class Reference

[Factory](#) class.#include <[Factory.h](#)>Inheritance diagram for [Factory](#):

## Public Member Functions

- [Factory](#) ([Type](#) \*type, [AddOn](#) \*addOn)  
*Instantiates the factory.*
- [~Factory](#) ()  
*Destroys the factory object.*
- virtual [Entity](#) \* [createEntity](#) ([Alliance](#) \*alliance)=0
- [Type](#) \* [getType](#) ()  
*Returns factories type state.*
- void [setType](#) ([Type](#) \*type)  
*Sets the factories type state.*
- [AddOn](#) \* [getAddOn](#) ()  
*Returns factories add ons.*
- void [setAddOns](#) ([AddOn](#) \*addOn)  
*Sets the factories add ons.*
- virtual [Factory](#) \* [clone](#) ()=0

### 4.13.1 Detailed Description

[Factory](#) class.

Used to instantiate [Entity](#) objects.

Definition at line 12 of file [Factory.h](#).

### 4.13.2 Constructor & Destructor Documentation

#### 4.13.2.1 Factory()

```
Factory::Factory (
    Type * type,
    AddOn * addOn )
```

Instantiates the factory.

##### Parameters

|              |                  |
|--------------|------------------|
| <i>type</i>  | must be a Type*  |
| <i>addOn</i> | must be a AddOn* |

Definition at line 3 of file [Factory.cpp](#).

```
00003                                     {
00004     this->type = type;
00005     this->addOn = addOn;
00006 }
```

#### 4.13.2.2 ~Factory()

```
Factory::~~Factory ( )
```

Destroys the factory object.

Postconditions:

- All dynamic memory should be deallocated from the factory object

Definition at line 8 of file [Factory.cpp](#).

```
00008     {
00009     delete type;
00010     delete addOn;
00011 }
```

### 4.13.3 Member Function Documentation

#### 4.13.3.1 clone()

```
virtual Factory * Factory::clone ( ) [pure virtual]
```

Implemented in [PersonnelFactory](#), [SupportFactory](#), and [VehicleFactory](#).

#### 4.13.3.2 createEntity()

```
virtual Entity * Factory::createEntity (
    Alliance * alliance ) [pure virtual]
```

Implemented in [PersonnelFactory](#), [SupportFactory](#), and [VehicleFactory](#).

#### 4.13.3.3 getAddOn()

```
AddOn * Factory::getAddOn ( )
```

Returns factories add ons.

Postconditions:

- Returns the add ons of the factory

##### Returns

[AddOn](#)\* The decorators for the factory object

Definition at line 21 of file [Factory.cpp](#).

```
00021 {
00022     return this->addOn;
00023 }
```

#### 4.13.3.4 getType()

```
Type * Factory::getType ( )
```

Returns factories type state.

Postconditions:

- Returns the type

##### Returns

[Type](#)\* The type state of the factory object

Definition at line 13 of file [Factory.cpp](#).

```
00013 {
00014     return this->type;
00015 }
```

#### 4.13.3.5 setAddOns()

```
void Factory::setAddOns (
    AddOn * addOn )
```

Sets the factories add ons.

Preconditions:

- addOns must be an AddOn\*

Postconditions:

- Sets the add ons of the factory object

##### Parameters

|              |                  |
|--------------|------------------|
| <i>addOn</i> | must be a AddOn* |
|--------------|------------------|

##### Returns

void

Definition at line 25 of file [Factory.cpp](#).

```
00025 {
00026     this->addOn = addOn;
00027 }
```

#### 4.13.3.6 setType()

```
void Factory::setType (
    Type * type )
```

Sets the factories type state.

Preconditions:

- type must be an Type\*

Postconditions:

- Sets the type state of the factory object

##### Parameters

|             |                 |
|-------------|-----------------|
| <i>type</i> | must be a Type* |
|-------------|-----------------|



**Returns**

void

Definition at line 17 of file [Factory.cpp](#).

```

00017
00018     this->type = type;
00019 }

```

The documentation for this class was generated from the following files:

- [Factory.h](#)
- [Factory.cpp](#)

## 4.14 General Class Reference

### Public Member Functions

- [General](#) ([Alliance](#) \*alliance, [Strategy](#) \*stratety)  
*Construct a new [General](#) object.*
- void [initiateStrategy](#) ([KeyPoint](#) \*keyPoint)  
*The function initiates the strategy.*
- bool [setStrategy](#) ([Strategy](#) \*strategy)  
*Set the [Strategy](#) object.*
- [Alliance](#) \* [getAlliance](#) ()  
*Returns the [Alliance](#) object.*

#### 4.14.1 Detailed Description

Definition at line 8 of file [General.h](#).

#### 4.14.2 Constructor & Destructor Documentation

##### 4.14.2.1 General()

```

General::General (
    Alliance * alliance,
    Strategy * stratety )

```

Construct a new [General](#) object.

**Parameters**

|                 |                      |
|-----------------|----------------------|
| <i>alliance</i> | must be an Alliance* |
| <i>stratety</i> | must be a Strategy*  |

Definition at line 3 of file [General.cpp](#).

```
00003
00004     this->alliance = alliance;
00005     this->strategy = strategy;
00006     numDeaths = 0;
00007 }
```

### 4.14.3 Member Function Documentation

#### 4.14.3.1 getAlliance()

[Alliance](#) \* General::getAlliance ( )

Returns the [Alliance](#) object.

PostConditions:

- Returns the alliance of the general

**Returns**

[Alliance](#)\* The alliance that the general is associated

Definition at line 22 of file [General.cpp](#).

```
00022
00023     return this->alliance;
00024 }
```

#### 4.14.3.2 initiateStrategy()

```
void General::initiateStrategy (
    KeyPoint * keyPoint )
```

The function initiates the strategy.

Precondition:

- keyPoint must be a [KeyPoint](#)\*

**Parameters**

|                 |                                      |
|-----------------|--------------------------------------|
| <i>keyPoint</i> | must be a <a href="#">KeyPoint</a> * |
|-----------------|--------------------------------------|

**Returns**

void

Definition at line 9 of file [General.cpp](#).

```
00009                                     {
00010     numDeaths++;
00011     if (numDeaths >= 5) {
00012         strategy->performStrat(keyPoint, this->alliance);
00013         numDeaths = 0;
00014     }
00015 }
```

**4.14.3.3 setStrategy()**

```
bool General::setStrategy (
    Strategy * strategy )
```

Set the [Strategy](#) object.

PreConditions:

- strategy must be of type Strategy\*

PostConditions:

- true is returned if setting the strategy was successful
- false is returned if setting the strategy was unsuccessful

**Parameters**

|                 |  |
|-----------------|--|
| <i>strategy</i> |  |
|-----------------|--|

**Returns**

true if the setting the [Strategy](#) object was successful  
false if the setting the [Strategy](#) object was unsuccessful

Definition at line 17 of file [General.cpp](#).

```
00017                                     {
00018     this->strategy = strategy;
00019     return true;
00020 }
```

The documentation for this class was generated from the following files:

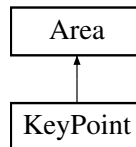
- General.h
- General.cpp

## 4.15 KeyPoint Class Reference

Keypoint class.

```
#include <KeyPoint.h>
```

Inheritance diagram for KeyPoint:



### Public Member Functions

- [KeyPoint](#) (std::string areaName)  
*Instantiates the key point.*
- [KeyPoint](#) ([KeyPoint](#) &keyPoint)  
*Instantiates a copy of a [KeyPoint](#).*
- bool [isKeyPoint](#) ()  
*Returns area type.*
- void [simulateBattle](#) ([Alliance](#) \*alliance)  
*Simulate Battle with troops from the alliance passed in.*
- void [clearBattlefield](#) ([Alliance](#) \*alliance)  
*Clears the battlefield of all deceased troops.*
- void [moveEntitiesInto](#) ([Alliance](#) \*alliance, int numTroops)  
*Moves a specific alliances troops into this keypoint.*
- void [moveEntitiesOutOf](#) ([Alliance](#) \*alliance, int numTroops)  
*Moves a specific alliances troops out of the keypoint.*
- void [addEntity](#) ([Entity](#) \*entity)  
*Adds an entity to the key point object.*
- void [addGeneral](#) ([General](#) \*general)
- void [removeGeneral](#) ([General](#) \*general)
- [Area](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current Keypoint.*
- void [changeWeather](#) ()  
*Switches the [Weather](#) object to the next state.*
- void [setWeather](#) ([Weather](#) \*weather)  
*Set the [Weather](#) object.*
- std::string [getWeather](#) () const  
*The weather at the current state is returned.*

### 4.15.1 Detailed Description

Keypoint class.

Used to emulate strategic positions.

Definition at line 17 of file [KeyPoint.h](#).

## 4.15.2 Constructor & Destructor Documentation

### 4.15.2.1 KeyPoint() [1/2]

```
KeyPoint::KeyPoint (
    std::string areaName )
```

Instantiates the key point.

#### Parameters

|                 |                   |
|-----------------|-------------------|
| <i>areaName</i> | must be an string |
|-----------------|-------------------|

### 4.15.2.2 KeyPoint() [2/2]

```
KeyPoint::KeyPoint (
    KeyPoint & keyPoint )
```

Instantiates a copy of a [KeyPoint](#).

#### Parameters

|                 |  |
|-----------------|--|
| <i>keyPoint</i> | must be an <a href="#">KeyPoint</a> instance |
|-----------------|--|

Definition at line 15 of file [KeyPoint.cpp](#).

```
00015                                     : Area(keyPoint.getAreaName()) {
00016     for (int i = 0; i < keyPoint.entities.size(); i++)
00017         this->addEntity(keyPoint.entities[i]->clone());
00018
00019     weather = keyPoint.weather->clone();
00020 }
```

### 4.15.2.3 ~KeyPoint()

```
KeyPoint::~~KeyPoint ( )
```

Definition at line 22 of file [KeyPoint.cpp](#).

```
00022     {
00023     for (int i = 0; i < entities.size(); i++)
00024         delete entities[i];
00025
00026     for (int i = 0; i < generals.size(); i++)
00027         delete generals[i];
00028
00029     delete weather;
00030 }
```

### 4.15.3 Member Function Documentation

#### 4.15.3.1 addEntity()

```
void KeyPoint::addEntity (
    Entity * entity )
```

Adds an entity to the key point object.

Preconditions:

- entity must be an Entity\*

Postconditions:

- Add entity to key point

##### Parameters

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

##### Returns

void

Definition at line 113 of file [KeyPoint.cpp](#).

```
00113                                     {
00114     entities.push_back(entity);
00115 }
```

#### 4.15.3.2 addGeneral()

```
void KeyPoint::addGeneral (
    General * general ) [virtual]
```

Implements [Area](#).

Definition at line 117 of file [KeyPoint.cpp](#).

```
00117                                     {
00118     generals.push_back(general);
00119 }
```

### 4.15.3.3 changeWeather()

```
void KeyPoint::changeWeather ( )
```

Switches the [Weather](#) object to the next state.

Definition at line 140 of file [KeyPoint.cpp](#).

```
00140     {
00141
00142         srand(time(0));
00143
00144         int randomNum = 1 + (rand() % 10);
00145         std::string currWeather = this->weather->getWeather();
00146
00147         if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
            staying
00148             this->weather->handleChange(this);
00149         else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
            Cloudy and staying
00150             this->weather->handleChange(this);
00151         else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
            and staying
00152             this->weather->handleChange(this);
00153
00154
00155     }
```

### 4.15.3.4 clearBattlefield()

```
void KeyPoint::clearBattlefield (
    Alliance * alliance )
```

Clears the battlefield of all deceased troops.

Postconditions:

- Notify command centers of each troop who is killed

#### Parameters

|                 |                      |
|-----------------|----------------------|
| <i>alliance</i> | must be an Alliance* |
|-----------------|----------------------|

#### Returns

void

Definition at line 61 of file [KeyPoint.cpp](#).

```
00061     {
00062         int destroyed = 0;
00063         double numUnits = 0;
00064         for (vector<Entity*>::iterator it = entities.begin(); it != entities.end(); ++it) {
00065             if ((*it)->getHealth() <= 0) {
00066                 destroyed++;
00067                 for (int i = 0; i < generals.size(); i++) {
00068                     if (generals[i]->getAlliance() == (*it)->getAlliance()) {
00069                         generals[i]->initiateStrategy(this);
00070                         delete *it;
00071                         entities.erase(it);
00072                     }
00073                 }
00074             } else if ((*it)->getAlliance() == alliance) {
```

```

00075         numUnits++;
00076     }
00077 }
00078
00079 // saving stats
00080 string stats = getAreaName() + ":\n";
00081 stats += "Key Point Satus: ";
00082 if (numUnits / entities.size() >= 0.6) {
00083     stats += "Winning\n";
00084 } else if (numUnits / entities.size() >= 0.35) {
00085     stats += "Contested\n";
00086 } else {
00087     stats += "Losing\n";
00088 }
00089 stats += "Number of Entities Destroyed by Alliance: " + to_string(destroyed);
00090
00091 RoundStats::keyPointInformation.push_back(stats);
00092 RoundStats::numEntitiesDestroyed += destroyed;
00093 }

```

#### 4.15.3.5 clone()

`Area * KeyPoint::clone ( ) [virtual]`

Instantiates and returns a clone of the current Keypoint.

Postconditions:

- Returns the clone of the current Keypoint

#### Returns

Area\* The Keypoint clone

Implements [Area](#).

Definition at line 131 of file [KeyPoint.cpp](#).

```

00131     {
00132         return new KeyPoint (*this);
00133     }

```

#### 4.15.3.6 getWeather()

`std::string KeyPoint::getWeather ( ) const`

The weather at the current state is returned.

#### Returns

string The weather state

Definition at line 157 of file [KeyPoint.cpp](#).

```

00157     {
00158         return this->weather->getWeather();
00159     }

```



#### 4.15.3.7 isKeyPoint()

```
bool KeyPoint::isKeyPoint ( ) [virtual]
```

Returns area type.

Postconditions:

- Returns true

Returns

bool The area type

Implements [Area](#).

Definition at line 32 of file [KeyPoint.cpp](#).

```
00032     {  
00033         return true;  
00034     }
```

#### 4.15.3.8 moveEntitiesInto()

```
void KeyPoint::moveEntitiesInto (  
    Alliance * alliance,  
    int numTroops )
```

Moves a specific alliances troops into this keypoint.

Preconditions:

- alliance must be an Alliance\*
- numTroops must be an int

Postconditions:

- Move troops to into this keypoint

Parameters

|                  |                      |
|------------------|----------------------|
| <i>alliance</i>  | must be an Alliance* |
| <i>numTroops</i> | must be an int       |

Returns

void

Definition at line 95 of file [KeyPoint.cpp](#).

```
00095 {
00096     vector<Entity*> troops = alliance->getReserveEntities(numTroops);
00097     for (int i = 0; i < troops.size(); i++)
00098         entities.push_back(troops[i]);
00099 }
```

#### 4.15.3.9 moveEntitiesOutOf()

```
void KeyPoint::moveEntitiesOutOf (
    Alliance * alliance,
    int numTroops )
```

Moves a specific alliances troops out of the keypoint.

Preconditions:

- alliance must be an Alliance\*
- numTroops must be an int

Postconditions:

- Move troops to reserve

##### Parameters

|                  |                      |
|------------------|----------------------|
| <i>alliance</i>  | must be an Alliance* |
| <i>numTroops</i> | must be an int       |

##### Returns

void

Definition at line 101 of file [KeyPoint.cpp](#).

```
00101 {
00102     vector<Entity*>::iterator it = entities.begin();
00103     for (int i = 0; i < numTroops && it != entities.end(); i++) {
00104         for (; it != entities.end(); ++it) {
00105             if ((*it)->getAlliance() == alliance) {
00106                 alliance->addReserveEntity(*it);
00107                 entities.erase(it);
00108             }
00109         }
00110     }
00111 }
```

#### 4.15.3.10 removeGeneral()

```
void KeyPoint::removeGeneral (
    General * general )
```

Definition at line 121 of file [KeyPoint.cpp](#).

```
00121                                     {
00122     for (vector<General*>::iterator it = generals.begin(); it != generals.end(); ++it) {
00123         if (*it == general) {
00124             delete *it;
00125             generals.erase(it);
00126             return;
00127         }
00128     }
00129 }
```

#### 4.15.3.11 setWeather()

```
void KeyPoint::setWeather (
    Weather * weather )
```

Set the [Weather](#) object.

Preconditions:

- weather must be a Weather\*

Postconditions:

- must set the keyPoints weather state

##### Parameters

|                |                    |
|----------------|--------------------|
| <i>weather</i> | must be a Weather* |
|----------------|--------------------|

##### Returns

void

Definition at line 135 of file [KeyPoint.cpp](#).

```
00135                                     {
00136     delete this->weather;
00137     this->weather = weather;
00138 }
```

#### 4.15.3.12 simulateBattle()

```
void KeyPoint::simulateBattle (
    Alliance * alliance ) [virtual]
```

Simulate Battle with troops from the alliance passed in.

Preconditions:

- alliance must be an Alliance\*

Postconditions:

- Perform attacks on other alliance troops

## Parameters

|                 |                      |
|-----------------|----------------------|
| <i>alliance</i> | must be an Alliance* |
|-----------------|----------------------|

## Returns

void

Implements [Area](#).Definition at line 36 of file [KeyPoint.cpp](#).

```

00036                                     {
00037     int numUnits = 0;
00038     for (int i = 0; i < entities.size(); i++) {
00039         if (entities[i]->getAlliance() == alliance) {
00040             numUnits++;
00041         }
00042     }
00043
00044     if (numUnits != entities.size()) {
00045         for (int i = 0; i < entities.size(); i++) {
00046             if (entities[i]->getAlliance() == alliance) {
00047                 int random;
00048                 do {
00049                     random = rand() % entities.size();
00050                 } while (entities[random]->getAlliance() == alliance);
00051
00052                 if (rand() % (int)(weather->getMultiplier() * 100) <= (int)(weather->getMultiplier() *
100))
00053                     entities[i]->dealDamage(entities[random]);
00054             }
00055         }
00056     }
00057
00058     clearBattlefield(alliance);
00059 }

```

The documentation for this class was generated from the following files:

- [KeyPoint.h](#)
- [KeyPoint.cpp](#)

## 4.16 Negotiator Class Reference

### Public Member Functions

- [Negotiator](#) ()  
*Instantiates the [Negotiator](#).*
- [~Negotiator](#) ()  
*Destructor for the [Negotiator](#) object.*
- bool [sendPeace](#) ([Alliance](#) \*offerAlliance)  
*Tries to offer peace to all the alliances in vector.*
- void [removeAlliance](#) ([Alliance](#) \*oldAlliance)  
*Removes an alliance from the alliance vector.*
- void [addAlliance](#) ([Alliance](#) \*newAlliance)  
*Adds an alliance to the alliance vector.*
- int [getNumAlliances](#) ()  
*gets the number of Alliances in the negotiator*

### 4.16.1 Detailed Description

Definition at line 6 of file [Negotiator.h](#).

### 4.16.2 Constructor & Destructor Documentation

#### 4.16.2.1 Negotiator()

```
Negotiator::Negotiator ( )
```

Instantiates the [Negotiator](#).

Definition at line 4 of file [Negotiator.cpp](#).

```
00004 {}
```

#### 4.16.2.2 ~Negotiator()

```
Negotiator::~~Negotiator ( )
```

Destructor for the [Negotiator](#) object.

Definition at line 6 of file [Negotiator.cpp](#).

```
00006 {  
00007     alliances.clear();  
00008 }
```

### 4.16.3 Member Function Documentation

#### 4.16.3.1 addAlliance()

```
void Negotiator::addAlliance (  
    Alliance * newAlliance )
```

Adds an alliance to the alliance vector.

Preconditions:

- newAlliance must be an [Alliance](#) pointer

Postconditions:

- [Alliance](#) is added to the vector

Returns

void

Definition at line 34 of file [Negotiator.cpp](#).

```
00034 {  
00035  
00036     if (std::find(alliances.begin(), alliances.end(), newAlliance) != alliances.end())  
00037         alliances.push_back(newAlliance);  
00038  
00039 }
```

#### 4.16.3.2 getNumAlliances()

```
int Negotiator::getNumAlliances ( )
```

gets the number of Alliances in the negotiator

Postconditions:

- Returns the number of alliances

Returns

int Number of alliances

Definition at line 41 of file [Negotiator.cpp](#).

```
00041 {  
00042     return this->alliances.size();  
00043 }
```

#### 4.16.3.3 removeAlliance()

```
void Negotiator::removeAlliance (  
    Alliance * oldAlliance )
```

Removes an alliance from the alliance vector.

Preconditions:

- oldAlliance must be an [Alliance](#) pointer

Postconditions:

- [Alliance](#) is removed from vector

Returns

void

Definition at line 24 of file [Negotiator.cpp](#).

```
00024 {  
00025  
00026     for (int xx = 0; xx < alliances.size(); xx++)  
00027     {  
00028         if (alliances[xx]->getID() == oldAlliance->getID())  
00029             alliances.erase( alliances.begin() + xx ); // Removes the specific alliances from this  
negotiator  
00030     }  
00031  
00032 }
```

#### 4.16.3.4 sendPeace()

```
bool Negotiator::sendPeace (  
    Alliance * offerAlliance )
```

Tries to offer peace to all the alliances in vector.

Preconditions:

- offerAlliance must be an [Alliance](#) pointer

Postconditions:

- Iterates through alliance vector and calls considerPeace for the enemies

## Parameters

|           |                |
|-----------|----------------|
| <i>id</i> | must be an int |
|-----------|----------------|

## Returns

bool

Definition at line 10 of file [Negotiator.cpp](#).

```

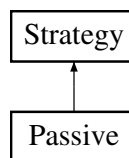
00010                                     {
00011
00012     for (int yy = 0; yy < alliances.size(); yy++)
00013     {
00014         if (alliances[yy] != offerAlliance) {
00015             if (alliances[yy]->considerPeace() == false)
00016                 return false; // There is at least one enemy alliances that does not want the peace
00017     deal
00017     }
00018
00019     }
00020
00021     return true; // All the alliances being fought against agreed to the peace deal
00022 }
```

The documentation for this class was generated from the following files:

- Negotiator.h
- Negotiator.cpp

## 4.17 Passive Class Reference

Inheritance diagram for Passive:



### Public Member Functions

- void [performStrat](#) ([KeyPoint](#) \*keyPoint, [Alliance](#) \*alliance)  
This function will perform a *Passive* strategy.
- [Strategy](#) \* [clone](#) ()  
Returns the clone of the *Passive Strategy* object.

### Additional Inherited Members

#### 4.17.1 Detailed Description

Definition at line 6 of file [Passive.h](#).

## 4.17.2 Constructor & Destructor Documentation

### 4.17.2.1 Passive()

```
Passive::Passive ( )
```

Definition at line 5 of file [Passive.cpp](#).

```
00005 {}
```

## 4.17.3 Member Function Documentation

### 4.17.3.1 clone()

```
Strategy * Passive::clone ( ) [virtual]
```

Returns the clone of the [Passive Strategy](#) object.

#### Returns

Strategy\* The clone of the [Passive Strategy](#) object

Implements [Strategy](#).

Definition at line 13 of file [Passive.cpp](#).

```
00013 {
00014     return new Passive();
00015 }
```

### 4.17.3.2 performStrat()

```
void Passive::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [virtual]
```

This function will perform a [Passive](#) strategy.

#### Author

Antwi-Antwi

#### Parameters

|                 |   |
|-----------------|---|
| <i>keyPoint</i> | a <a href="#">Passive</a> strategy will then be performed at this specific keypoint |
|-----------------|---|



**Returns**

void The function will return void

Implements [Strategy](#).

Definition at line 7 of file [Passive.cpp](#).

```
00007                                     {
00008
00009     int randomNumber = (rand() % 10) + 5;
00010     keyPoint->moveEntitiesOutOf(alliance, randomNumber);
00011 }
```

The documentation for this class was generated from the following files:

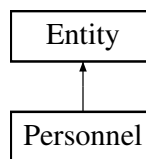
- [Passive.h](#)
- [Passive.cpp](#)

## 4.18 Personnel Class Reference

[Personnel](#) class.

```
#include <Personnel.h>
```

Inheritance diagram for [Personnel](#):



### Public Member Functions

- [Personnel](#) ([Type](#) \*type, int health=100, int damage=10)  
*Instantiates the [Personnel](#).*
- void [takeDamage](#) (int damage)  
*Removes health from the [Personnel](#) object.*
- void [dealDamage](#) ([Entity](#) \*entity)  
*Inflicts damage onto another entity.*
- [Entity](#) \* [clone](#) ()  
*Returns the clone of the [Personnel](#) object.*

### 4.18.1 Detailed Description

[Personnel](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 11 of file [Personnel.h](#).

## 4.18.2 Constructor & Destructor Documentation

### 4.18.2.1 Personnel()

```
Personnel::Personnel (
    Type * type,
    int health = 100,
    int damage = 10 )
```

Instantiates the [Personnel](#).

#### Parameters

|               |                 |
|---------------|-----------------|
| <i>health</i> | must be an int  |
| <i>damage</i> | must be an int  |
| <i>type</i>   | must be a Type* |

Definition at line 5 of file [Personnel.cpp](#).

```
00005 : Entity(type, health, damage) {}
```

## 4.18.3 Member Function Documentation

### 4.18.3.1 clone()

```
Entity * Personnel::clone ( ) [virtual]
```

Returns the clone of the [Personnel](#) object.

#### Returns

Entity\* Clone of personnel object

Implements [Entity](#).

Definition at line 16 of file [Personnel.cpp](#).

```
00016 {
00017     Personnel* p;
00018     if (this->getType() == NULL) {
00019         p = new Personnel(NULL, this->getHealth(), this->getDamage());
00020     } else {
00021         p = new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
00022     }
00023     p->setAlliance(this->getAlliance());
00024     return p;
00025 }
00026
00027 }
```

#### 4.18.3.2 dealDamage()

```
void Personnel::dealDamage (
    Entity * entity ) [virtual]
```

Inflicts damage onto another entity.

Preconditions:

- entity must be an Entity\*

Postconditions:

- Reduces the health of the entity

Parameters

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

Returns

void

Implements [Entity](#).

Definition at line 11 of file [Personnel.cpp](#).

```
00011 {
00012     RoundStats::damageDone += getDamage();
00013     entity->takeDamage(getDamage());
00014 }
```

#### 4.18.3.3 takeDamage()

```
void Personnel::takeDamage (
    int damage ) [virtual]
```

Removes health from the [Personnel](#) object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the [Personnel](#) object

## Parameters

|               |                |
|---------------|----------------|
| <i>damage</i> | must be an int |
|---------------|----------------|

## Returns

void

Implements [Entity](#).

Definition at line 7 of file [Personnel.cpp](#).

```
00007 {
00008     setHealth(getHealth() - damage);
00009 }
```

The documentation for this class was generated from the following files:

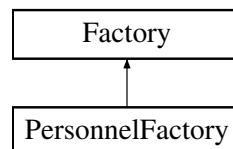
- [Personnel.h](#)
- [Personnel.cpp](#)

## 4.19 PersonnelFactory Class Reference

[PersonnelFactory](#) class.

```
#include <PersonnelFactory.h>
```

Inheritance diagram for [PersonnelFactory](#):



### Public Member Functions

- [PersonnelFactory](#) ([Type](#) \*type, [AddOn](#) \*addOn)  
*Instantiates the [Personnel](#) factory.*
- [Entity](#) \* [createEntity](#) ([Alliance](#) \*alliance)  
*Instantiates and returns a [Personnel](#) for the given alliance.*
- [Factory](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current [Personnel](#) factory.*

#### 4.19.1 Detailed Description

[PersonnelFactory](#) class.

Used to instantiate [Personnel](#) objects.

Definition at line 11 of file [PersonnelFactory.h](#).

## 4.19.2 Constructor & Destructor Documentation

### 4.19.2.1 PersonnelFactory()

```
PersonnelFactory::PersonnelFactory (
    Type * type,
    AddOn * addOn )
```

Instantiates the [Personnel](#) factory.

#### Parameters

|              |                  |
|--------------|------------------|
| <i>type</i>  | must be a Type*  |
| <i>addOn</i> | must be a AddOn* |

Definition at line 5 of file [PersonnelFactory.cpp](#).

```
00005 : Factory(type, addOn) {}
```

## 4.19.3 Member Function Documentation

### 4.19.3.1 clone()

```
Factory * PersonnelFactory::clone ( ) [virtual]
```

Instantiates and returns a clone of the current [Personnel](#) factory.

Postconditions:

- Returns the clone of the current [Personnel](#) factory

#### Returns

Factory\* The [Personnel](#) factory clone

Implements [Factory](#).

Definition at line 19 of file [PersonnelFactory.cpp](#).

```
00019         {
00020     return new PersonnelFactory(getType()->clone(), getAddOn()->clone());
00021 }
```

#### 4.19.3.2 createEntity()

```
Entity * PersonnelFactory::createEntity (
    Alliance * alliance ) [virtual]
```

Instantiates and returns a [Personnel](#) for the given alliance.

Preconditions:

- alliance must be an Alliance\*

Postconditions:

- Returns the instantiated [Personnel](#) object with specific state

##### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>alliance</i> | must be a Alliance* |
|-----------------|---------------------|

##### Returns

Entity\* The instatiated personnel

Implements [Factory](#).

Definition at line 7 of file [PersonnelFactory.cpp](#).

```
00007 {
00008     Personnel* p = new Personnel(getType()->clone());
00009     p->setAlliance(alliance);
00010     if (getAddOn() != NULL) {
00011         AddOn* personnelAddOn = getAddOn()->clone();
00012         personnelAddOn->setEntity(p);
00013         return personnelAddOn;
00014     } else {
00015         return p;
00016     }
00017 }
```

The documentation for this class was generated from the following files:

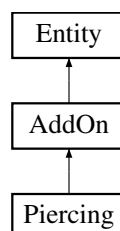
- PersonnelFactory.h
- PersonnelFactory.cpp

## 4.20 Piercing Class Reference

[Piercing](#) class.

```
#include <Piercing.h>
```

Inheritance diagram for Piercing:



## Public Member Functions

- [Piercing](#) (int value)  
*Instantiates an [Piercing](#).*
- void [takeDamage](#) (int damage)  
*Decreases the entities' armour value (or health when their armour has depleted)*
- void [dealDamage](#) ([Entity](#) \*entity)  
*Adds to the damage [Entity](#) objects inflict.*
- [AddOn](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current [Piercing](#).*

## Additional Inherited Members

### 4.20.1 Detailed Description

[Piercing](#) class.

Used to add to the damage [Entity](#) objects inflict.

Definition at line 11 of file [Piercing.h](#).

### 4.20.2 Constructor & Destructor Documentation

#### 4.20.2.1 Piercing()

```
Piercing::Piercing (
    int value )
```

Instantiates an [Piercing](#).

#### Parameters

|              |                |
|--------------|----------------|
| <i>value</i> | must be an int |
|--------------|----------------|

Definition at line 4 of file [Piercing.cpp](#).  
00004 : [AddOn](#)(value) {}

### 4.20.3 Member Function Documentation

#### 4.20.3.1 clone()

```
AddOn * Piercing::clone ( ) [virtual]
```

Instantiates and returns a clone of the current [Piercing](#).

Postconditions:

- Returns the clone of the current [Piercing](#)

**Returns**

Piercing\* The [Piercing](#) clone

Implements [AddOn](#).

Definition at line 16 of file [Piercing.cpp](#).

```
00016         {
00017     Piercing* piercing = new Piercing(value);
00018     if (getEntity() != NULL)
00019         piercing->setEntity(entity->clone());
00020     return piercing;
00021 }
```

#### 4.20.3.2 dealDamage()

```
void Piercing::dealDamage (
    Entity * entity ) [virtual]
```

Adds to the damage [Entity](#) objects inflict.

Preconditions:

- entity must be an Entity\*

Postconditions:

- Inflicts damage to passed in [Entity](#) objects using the sum of it's value and the entity onto which it has been added's value

**Parameters**

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

**Returns**

void

Implements [AddOn](#).

Definition at line 10 of file [Piercing.cpp](#).

```
00010         {
00011     int sumValue = this->entity->getDamage() + value;
00012     entity->takeDamage(sumValue);
00013     RoundStats::damageDone += sumValue;
00014 }
```



### 4.20.3.3 takeDamage()

```
void Piercing::takeDamage (
    int damage ) [virtual]
```

Decreases the entities' armour value (or health when their armour has depleted)

Preconditions:

- damage must be an int

Postconditions:

- Does nothing

#### Parameters

|               |                |
|---------------|----------------|
| <i>damage</i> | must be an int |
|---------------|----------------|

#### Returns

void

Implements [AddOn](#).

Definition at line 6 of file [Piercing.cpp](#).

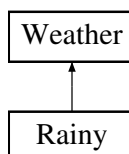
```
00006 {
00007     entity->takeDamage (damage);
00008 }
```

The documentation for this class was generated from the following files:

- Piercing.h
- Piercing.cpp

## 4.21 Rainy Class Reference

Inheritance diagram for Rainy:



## Public Member Functions

- [Rainy](#) ()  
*Instantiates the Runny object of the state pattern.*
- `std::string` [getWeather](#) ()  
*Returns string which tels us the weather.*
- `void` [handleChange](#) ([KeyPoint](#) \*k)  
*Will change the current state of the weather inside the specific keypoint.*
- [Weather](#) \* [clone](#) ()  
*Returns a clone of the [Rainy](#) object.*

## Additional Inherited Members

### 4.21.1 Detailed Description

Definition at line 6 of file [Rainy.h](#).

### 4.21.2 Constructor & Destructor Documentation

#### 4.21.2.1 Rainy()

```
Rainy::Rainy ( )
```

Instantiates the Runny object of the state pattern.

Definition at line 4 of file [Rainy.cpp](#).

```
00004      : Weather() {  
00005      this->multiplier = 0.5;  
00006 }
```

### 4.21.3 Member Function Documentation

#### 4.21.3.1 clone()

```
Weather * Rainy::clone ( ) [virtual]
```

Returns a clone of the [Rainy](#) object.

#### Returns

`Weather*` The clone of the rainy object

Implements [Weather](#).

Definition at line 17 of file [Rainy.cpp](#).

```
00017      {  
00018      return new Rainy();  
00019 }
```

### 4.21.3.2 `getWeather()`

```
std::string Rainy::getWeather ( ) [virtual]
```

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

Returns

std::string which is the current state

Implements [Weather](#).

Definition at line 8 of file [Rainy.cpp](#).

```
00008 {  
00009     return "Rainy";  
00010 }
```

### 4.21.3.3 `handleChange()`

```
void Rainy::handleChange (  
    KeyPoint * k ) [virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- k must be a KeyPoint\*

Postconditions:

- Changes the current weather to the next one in the state pattern ([Sunny](#))

Parameters

|          |                     |
|----------|---------------------|
| <i>k</i> | must be a KeyPoint* |
|----------|---------------------|

Returns

void

Implements [Weather](#).

Definition at line 12 of file [Rainy.cpp](#).

```
00012
00013     Sunny* newWeather = new Sunny();
00014     k->setWeather(newWeather);
00015 }
```

The documentation for this class was generated from the following files:

- Rainy.h
- Rainy.cpp

## 4.22 RoundStats Class Reference

### Static Public Member Functions

- static void [clearStats](#) ()
- static string [toString](#) ()

### Static Public Attributes

- static int [numEntitiesCreated](#) = 0
- static int [numEntitiesDestroyed](#) = 0
- static int [damageDone](#) = 0
- static vector< string > [keyPointInformation](#)
- static vector< string > [entityMovementInformation](#)

#### 4.22.1 Detailed Description

Definition at line 9 of file [RoundStats.h](#).

#### 4.22.2 Member Function Documentation

##### 4.22.2.1 clearStats()

```
void RoundStats::clearStats ( ) [static]
```

Definition at line 9 of file [RoundStats.cpp](#).

```
00009     {
00010         numEntitiesCreated = 0;
00011         numEntitiesDestroyed = 0;
00012         keyPointInformation.clear();
00013         entityMovementInformation.clear();
00014     }
```

#### 4.22.2.2 toString()

```
string RoundStats::toString ( ) [static]
```

Definition at line 16 of file [RoundStats.cpp](#).

```
00016         {
00017             string out = "Number of Entities Created: " + to_string(numEntitiesCreated) + "\n";
00018             out += "Number of Entities Destroyed by Alliance: " + to_string(numEntitiesDestroyed) + "\n";
00019             out += "Damage Given by Alliance: " + to_string(damageDone) + "\n";
00020
00021             out += "\nKey Point Round Information:\n";
00022             for (int i = 0; i < keyPointInformation.size(); i++)
00023                 out += keyPointInformation[i] + "\n";
00024
00025             out += "\nMovement Round Information:\n";
00026             for (int i = 0; i < entityMovementInformation.size(); i++)
00027                 out += entityMovementInformation[i] + "\n";
00028
00029             return out;
00030 }
```

### 4.22.3 Member Data Documentation

#### 4.22.3.1 damageDone

```
int RoundStats::damageDone = 0 [static]
```

Definition at line 13 of file [RoundStats.h](#).

#### 4.22.3.2 entityMovementInformation

```
vector< string > RoundStats::entityMovementInformation [static]
```

Definition at line 15 of file [RoundStats.h](#).

#### 4.22.3.3 keyPointInformation

```
vector< string > RoundStats::keyPointInformation [static]
```

Definition at line 14 of file [RoundStats.h](#).

#### 4.22.3.4 numEntitiesCreated

```
int RoundStats::numEntitiesCreated = 0 [static]
```

Definition at line 11 of file [RoundStats.h](#).

#### 4.22.3.5 numEntitiesDestroyed

```
int RoundStats::numEntitiesDestroyed = 0 [static]
```

Definition at line 12 of file [RoundStats.h](#).

The documentation for this class was generated from the following files:

- RoundStats.h
- RoundStats.cpp

### 4.23 SaveArchive Class Reference

Stores a list of mementos containing simulation state.

```
#include <SaveArchive.h>
```

#### Public Member Functions

- [SaveArchive](#) ()  
*Instantiates the [SaveArchive](#) class.*
- void [addNewSave](#) (std::string newSaveName, [WarEngineMemento](#) \*newSave)  
*Adds a new save to the list of stored mementos.*
- [WarEngineMemento](#) \* [getLastSave](#) ()  
*Returns the last saved memento.*
- [WarEngineMemento](#) \* [getSave](#) (std::string name)  
*Returns the last saved memento. Preconditions:*
- void [clearSaveList](#) ()  
*Erases all saved mementos from the list of saves. Postconditions:*
- void [deleteSave](#) (std::string name)  
*Deletes a memento with the matching given name from the list of saved mementos. Preconditions:*

#### 4.23.1 Detailed Description

Stores a list of mementos containing simulation state.

Definition at line 11 of file [SaveArchive.h](#).

#### 4.23.2 Constructor & Destructor Documentation

#### 4.23.2.1 SaveArchive()

```
SaveArchive::SaveArchive ( )
```

Instantiates the [SaveArchive](#) class.

Definition at line 3 of file [SaveArchive.cpp](#).

```
00003 {}
```

### 4.23.3 Member Function Documentation

#### 4.23.3.1 addNewSave()

```
void SaveArchive::addNewSave (
    std::string newSaveName,
    WarEngineMemento * newSave )
```

Adds a new save to the list of stored mementos.

Preconditions:

- newSave must be a WarEngineMemento\*
- newSaveName must be a string

Postconditions:

- Adds a new memento to list of saves

Parameters

|                    |                             |
|--------------------|-----------------------------|
| <i>newSave</i>     | must be a WarEngineMemento* |
| <i>newSaveName</i> | must be a string            |

Returns

void

Definition at line 5 of file [SaveArchive.cpp](#).

```
00005
00006     saveList.insert({newSaveName, newSave});
00007 }
```

```
{
```

#### 4.23.3.2 clearSaveList()

```
void SaveArchive::clearSaveList ( )
```

Erases all saved mementos from the list of saves. Postconditions:

- Clears all elements in the saveList vector

#### Returns

void

Definition at line 35 of file [SaveArchive.cpp](#).

```
00035     {
00036         saveList.clear();
00037     }
```

#### 4.23.3.3 deleteSave()

```
void SaveArchive::deleteSave (
    std::string name )
```

Deletes a memento with the matching given name from the list of saved mementos. Preconditions:

- name must be a string in date/time format

Postconditions:

- Removes the element in the saveList vector with a name matching that of the parameter

#### Parameters

|             |          |
|-------------|----------|
| <i>name</i> | a string |
|-------------|----------|

#### Returns

void

#### Exceptions

|                          |                       |
|--------------------------|-----------------------|
| <i>std::out_of_range</i> | save archive is empty |
|--------------------------|-----------------------|

Definition at line 39 of file [SaveArchive.cpp](#).

```
00039     {
00040         if(saveList.size() == 0){
00041             std::__throw_out_of_range("Save archive is empty");
00042         }
00043
00044         auto iter = saveList.find(name) ;
00045
00046         if(iter == saveList.end())
00047             return;
00048
00049         saveList.erase( iter );
00050     }
```



#### 4.23.3.4 getLastSave()

`WarEngineMemento * SaveArchive::getLastSave ( )`

Returns the last saved memento.

Postconditions:

- Returns the last element in the saveList vector

Returns

WarEngineMemento\*

Exceptions

|                                    |   |
|------------------------------------|---|
| <code>std::out_of_range</code>     | save archive is empty                                 |
| <code>std::invalid_argument</code> | memento with given name is not found in memento list. |

Definition at line 9 of file [SaveArchive.cpp](#).

```

00009                                     {
00010
00011     if(saveList.size() == 0){
00012         throw "Save archive is empty.";
00013     }
00014
00015     WarEngineMemento* lastSave = saveList.begin()->second;
00016
00017     saveList.erase( saveList.begin() );
00018
00019     return lastSave;
00020 }
```

#### 4.23.3.5 getSave()

`WarEngineMemento * SaveArchive::getSave (`  
     `std::string name )`

Returns the last saved memento. Preconditions:

- name must be a string

Postconditions:

- Returns the last element in the saveList vector

Parameters

|             |          |
|-------------|----------|
| <i>name</i> | a string |
|-------------|----------|

**Returns**

WarEngineMemento\*

**Exceptions**

|                                |                       |
|--------------------------------|-----------------------|
| <code>std::out_of_range</code> | save archive is empty |
|--------------------------------|-----------------------|

Definition at line 22 of file [SaveArchive.cpp](#).

```

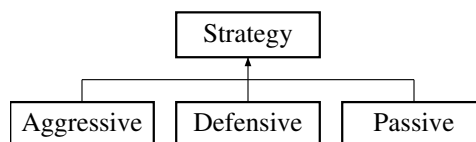
00022                                     {
00023     if(saveList.size() == 0){
00024         std::__throw_out_of_range("Save archive is empty");
00025     }
00026
00027     auto iter = saveList.find(name);
00028
00029     if(iter == saveList.end())
00030         std::__throw_invalid_argument("No save with given name exists");
00031
00032     return iter->second;
00033 }
```

The documentation for this class was generated from the following files:

- [SaveArchive.h](#)
- [SaveArchive.cpp](#)

## 4.24 Strategy Class Reference

Inheritance diagram for Strategy:

**Public Member Functions**

- [Strategy](#) ()  
*Construct a new [Strategy](#) object.*
- [~Strategy](#) ()  
*Destroy the [Strategy](#) object.*
- virtual void [performStrat](#) ([KeyPoint](#) \*keyPoint, [Alliance](#) \*alliance)=0
- virtual [Strategy](#) \* [clone](#) ()=0

**Protected Attributes**

- std::string [strategy](#)

### 4.24.1 Detailed Description

Definition at line 10 of file [Strategy.h](#).

## 4.24.2 Constructor & Destructor Documentation

### 4.24.2.1 Strategy()

```
Strategy::Strategy ( )
```

Construct a new [Strategy](#) object.

Definition at line 7 of file [Strategy.cpp](#).  
00007 {}

### 4.24.2.2 ~Strategy()

```
Strategy::~~Strategy ( )
```

Destroy the [Strategy](#) object.

Definition at line 9 of file [Strategy.cpp](#).  
00009 {}

## 4.24.3 Member Function Documentation

### 4.24.3.1 clone()

```
virtual Strategy * Strategy::clone ( ) [pure virtual]
```

Implemented in [Aggressive](#), [Defensive](#), and [Passive](#).

### 4.24.3.2 performStrat()

```
virtual void Strategy::performStrat (
    KeyPoint * keyPoint,
    Alliance * alliance ) [pure virtual]
```

Implemented in [Aggressive](#), [Defensive](#), and [Passive](#).

## 4.24.4 Member Data Documentation

#### 4.24.4.1 strategy

```
std::string Strategy::strategy [protected]
```

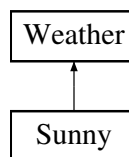
Definition at line 13 of file [Strategy.h](#).

The documentation for this class was generated from the following files:

- [Strategy.h](#)
- [Strategy.cpp](#)

## 4.25 Sunny Class Reference

Inheritance diagram for Sunny:



### Public Member Functions

- [Sunny](#) ()  
*Instantiates the [Sunny](#) object of the state pattern.*
- virtual std::string [getWeather](#) ()  
*Returns string which tells us the weather.*
- virtual void [handleChange](#) ([KeyPoint](#) \*k)  
*Will change the current state of the weather inside the specific keypoint.*
- [Weather](#) \* [clone](#) ()  
*Returns the clone of the [Sunny](#) object.*

### Additional Inherited Members

#### 4.25.1 Detailed Description

Definition at line 8 of file [Sunny.h](#).

#### 4.25.2 Constructor & Destructor Documentation

### 4.25.2.1 Sunny()

```
Sunny::Sunny ( )
```

Instantiates the [Sunny](#) object of the state pattern.

Definition at line 4 of file [Sunny.cpp](#).

```
00004         {  
00005             this->multiplier = 1.0;  
00006         }
```

## 4.25.3 Member Function Documentation

### 4.25.3.1 clone()

```
Weather * Sunny::clone ( ) [virtual]
```

Returns the clone of the [Sunny](#) object.

#### Returns

[Weather](#)\* Returns the clone of the sunny object

Implements [Weather](#).

Definition at line 17 of file [Sunny.cpp](#).

```
00017         {  
00018             return new Sunny();  
00019         }
```

### 4.25.3.2 getWeather()

```
std::string Sunny::getWeather ( ) [virtual]
```

Returns string which tells us the weather.

Postconditions:

- Returns the wether of ths current state

#### Returns

std::string which is the current state

Implements [Weather](#).

Definition at line 8 of file [Sunny.cpp](#).

```
00008         {  
00009             return "Sunny";  
00010         }
```

#### 4.25.3.3 `handleChange()`

```
void Sunny::handleChange (
    KeyPoint * k ) [virtual]
```

Will change the current state of the weather inside the specific keypoint.

Preconditions:

- `k` must be a `KeyPoint*`

Postconditions:

- Changes the current weather to the next one in the state pattern ([Cloudy](#))

##### Parameters

|                |                                  |
|----------------|----------------------------------|
| <code>k</code> | must be a <code>KeyPoint*</code> |
|----------------|----------------------------------|

##### Returns

`void`

Implements [Weather](#).

Definition at line 12 of file [Sunny.cpp](#).

```
00012 {
00013     Cloudy* newWeather = new Cloudy();
00014     k->setWeather(newWeather);
00015 }
```

The documentation for this class was generated from the following files:

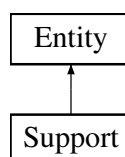
- `Sunny.h`
- `Sunny.cpp`

## 4.26 Support Class Reference

[Support](#) class.

```
#include <Support.h>
```

Inheritance diagram for `Support`:



## Public Member Functions

- [Support](#) ([Type](#) \*type, int health=1000, int damage=30)  
*Instantiates the support.*
- void [takeDamage](#) (int damage)  
*Removes health from the support object.*
- void [dealDamage](#) ([Entity](#) \*entity)  
*Inflicts damage onto another entity.*
- [Entity](#) \* [clone](#) ()  
*Returns the clone of the [Support](#) object.*

### 4.26.1 Detailed Description

[Support](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 11 of file [Support.h](#).

### 4.26.2 Constructor & Destructor Documentation

#### 4.26.2.1 Support()

```
Support::Support (
    Type * type,
    int health = 1000,
    int damage = 30 )
```

Instantiates the support.

##### Parameters

|               |                 |
|---------------|-----------------|
| <i>health</i> | must be an int  |
| <i>damage</i> | must be an int  |
| <i>type</i>   | must be a Type* |

Definition at line 4 of file [Support.cpp](#).

```
00004 : Entity(type, health, damage) {}
```

### 4.26.3 Member Function Documentation

#### 4.26.3.1 clone()

```
Entity * Support::clone ( ) [virtual]
```

Returns the clone of the [Support](#) object.

##### Returns

Entity\* The clone of the support object

Implements [Entity](#).

Definition at line 15 of file [Support.cpp](#).

```
00015         {
00016             Support* s;
00017             if (this->getType() == NULL) {
00018                 s = new Support(NULL, this->getHealth(), this->getDamage());
00019             } else {
00020                 s = new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
00021             }
00022             s->setAlliance(this->getAlliance());
00023             return s;
00024         }
00025     }
00026 }
```

#### 4.26.3.2 dealDamage()

```
void Support::dealDamage (
    Entity * entity ) [virtual]
```

Inflicts damage onto another entity.

##### Preconditions:

- entity must be an Entity\*

##### Postconditions:

- Reduces the health of the entity

##### Parameters

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

##### Returns

void

Implements [Entity](#).

Definition at line 6 of file [Support.cpp](#).



```

00006      {
00007      RoundStats::damageDone += getDamage();
00008      entity->takeDamage(getDamage());
00009  }

```

### 4.26.3.3 takeDamage()

```

void Support::takeDamage (
    int damage ) [virtual]

```

Removes health from the support object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the support object

#### Parameters

|               |                |
|---------------|----------------|
| <i>damage</i> | must be an int |
|---------------|----------------|

#### Returns

void

Implements [Entity](#).

Definition at line 11 of file [Support.cpp](#).

```

00011      {
00012      this->setHealth(this->getHealth() - damage);
00013  }

```

The documentation for this class was generated from the following files:

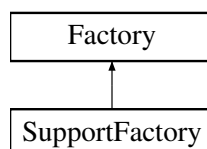
- Support.h
- Support.cpp

## 4.27 SupportFactory Class Reference

[SupportFactory](#) class.

```
#include <SupportFactory.h>
```

Inheritance diagram for SupportFactory:



## Public Member Functions

- [SupportFactory](#) ([Type](#) \*type, [AddOn](#) \*addOn)  
*Instantiates the support factory.*
- [Entity](#) \* [createEntity](#) ([Alliance](#) \*alliance)  
*Instantiates and returns a support for the given alliance.*
- [Factory](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current support factory.*

### 4.27.1 Detailed Description

[SupportFactory](#) class.

Used to instantiate [Support](#) objects.

Definition at line 11 of file [SupportFactory.h](#).

### 4.27.2 Constructor & Destructor Documentation

#### 4.27.2.1 SupportFactory()

```
SupportFactory::SupportFactory (
    Type * type,
    AddOn * addOn )
```

Instantiates the support factory.

#### Parameters

|              |                                   |
|--------------|-----------------------------------|
| <i>type</i>  | must be a <a href="#">Type</a> *  |
| <i>addOn</i> | must be a <a href="#">AddOn</a> * |

Definition at line 4 of file [SupportFactory.cpp](#).  
00004 : [Factory](#)(type, addOn) {}

### 4.27.3 Member Function Documentation

#### 4.27.3.1 clone()

```
Factory * SupportFactory::clone ( ) [virtual]
```

Instantiates and returns a clone of the current support factory.

Postconditions:

- Returns the clone of the current support factory

#### Returns

Factory\* The support factory clone

Implements [Factory](#).

Definition at line 18 of file [SupportFactory.cpp](#).

```
00018 {
00019     return new SupportFactory(getType()->clone(), getAddOn()->clone());
00020 }
```

#### 4.27.3.2 createEntity()

```
Entity * SupportFactory::createEntity (
    Alliance * alliance ) [virtual]
```

Instantiates and returns a support for the given alliance.

Preconditions:

- alliance must be an Alliance\*

Postconditions:

- Returns the instantiated support object with specific state

#### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>alliance</i> | must be a Alliance* |
|-----------------|---------------------|

#### Returns

Entity\* The instantiated support

Implements [Factory](#).

Definition at line 6 of file [SupportFactory.cpp](#).

```
00006 {
00007     Support* s = new Support(getType()->clone());
00008     s->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(s);
00012         return personnelAddOn;
00013     } else {
00014         return s;
00015     }
00016 }
```

The documentation for this class was generated from the following files:

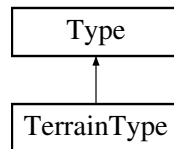
- SupportFactory.h
- SupportFactory.cpp

## 4.28 TerrainType Class Reference

[TerrainType](#) class.

```
#include <TerrainType.h>
```

Inheritance diagram for [TerrainType](#):



### Public Member Functions

- [TerrainType](#) ()  
*Instantiates the terrain type.*
- string [getTypeDesc](#) ()  
*Returns terrain type description.*
- [Type](#) \* [clone](#) ()

### 4.28.1 Detailed Description

[TerrainType](#) class.

Used to define [Entity](#) objects as terrain type.

Definition at line 11 of file [TerrainType.h](#).

### 4.28.2 Constructor & Destructor Documentation

#### 4.28.2.1 TerrainType()

```
TerrainType::TerrainType ( )
```

Instantiates the terrain type.

Definition at line 3 of file [TerrainType.cpp](#).

```
00003 {}
```

### 4.28.3 Member Function Documentation

### 4.28.3.1 clone()

```
Type * TerrainType::clone ( ) [virtual]
```

Implements [Type](#).

Definition at line 9 of file [TerrainType.cpp](#).

```
00009         {
00010         return new TerrainType();
00011     }
```

### 4.28.3.2 getTypeDesc()

```
string TerrainType::getTypeDesc ( ) [virtual]
```

Returns terrain type description.

Postconditions:

- Returns the terrain type

**Returns**

string The terrain type string

Implements [Type](#).

Definition at line 5 of file [TerrainType.cpp](#).

```
00005     {
00006     return "Terrain";
00007 }
```

The documentation for this class was generated from the following files:

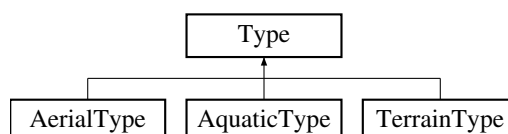
- TerrainType.h
- TerrainType.cpp

## 4.29 Type Class Reference

[Type](#) class.

```
#include <Type.h>
```

Inheritance diagram for [Type](#):



## Public Member Functions

- [Type](#) ()  
*Instantiates the type.*
- virtual string [getTypeDesc](#) ()=0
- virtual [Type](#) \* [clone](#) ()=0

### 4.29.1 Detailed Description

[Type](#) class.

Used to define [Entity](#) objects type.

Definition at line 13 of file [Type.h](#).

### 4.29.2 Constructor & Destructor Documentation

#### 4.29.2.1 Type()

```
Type::Type ( )
```

Instantiates the type.

Definition at line 3 of file [Type.cpp](#).  
00003 {}

### 4.29.3 Member Function Documentation

#### 4.29.3.1 getTypeDesc()

```
virtual string Type::getTypeDesc ( ) [pure virtual]
```

Implemented in [AerialType](#), [AquaticType](#), and [TerrainType](#).

The documentation for this class was generated from the following files:

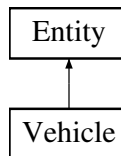
- [Type.h](#)
- [Type.cpp](#)

## 4.30 Vehicle Class Reference

[Vehicle](#) class.

```
#include <Vehicle.h>
```

Inheritance diagram for [Vehicle](#):



### Public Member Functions

- [Vehicle](#) ([Type](#) \*type, int health=500, int damage=10)  
*Instantiates the vehicle.*
- void [takeDamage](#) (int damage)  
*Removes health from the vehicle object.*
- void [dealDamage](#) ([Entity](#) \*entity)  
*Inflicts damage onto another entity.*
- [Entity](#) \* [clone](#) ()  
*Returns the clone of the [Vehicle](#) object.*

### 4.30.1 Detailed Description

[Vehicle](#) class.

Used to add additional functionality to [Entity](#) objects.

Definition at line 11 of file [Vehicle.h](#).

### 4.30.2 Constructor & Destructor Documentation

#### 4.30.2.1 [Vehicle](#)()

```
Vehicle::Vehicle (  
    Type * type,  
    int health = 500,  
    int damage = 10 )
```

Instantiates the vehicle.

## Parameters

|               |                 |
|---------------|-----------------|
| <i>health</i> | must be an int  |
| <i>damage</i> | must be an int  |
| <i>type</i>   | must be a Type* |

Definition at line 4 of file [Vehicle.cpp](#).

```
00004 : Entity(type, health, damage) {}
```

### 4.30.3 Member Function Documentation

#### 4.30.3.1 clone()

```
Entity * Vehicle::clone ( ) [virtual]
```

Returns the clone of the [Vehicle](#) object.

## Returns

Entity\* The clone of the vehicle object

Implements [Entity](#).

Definition at line 15 of file [Vehicle.cpp](#).

```
00015 {
00016     Vehicle* v;
00017     if (this->getType() == NULL) {
00018         v = new Vehicle(NULL, this->getHealth(), this->getDamage());
00019     } else {
00020         v = new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
00021     }
00022
00023     v->setAlliance(this->getAlliance());
00024
00025     return v;
00026 }
```

#### 4.30.3.2 dealDamage()

```
void Vehicle::dealDamage (
    Entity * entity ) [virtual]
```

Inflicts damage onto another entity.

## Preconditions:

- entity must be an Entity\*

## Postconditions:

- Reduces the health of the entity



## Parameters

|               |                    |
|---------------|--------------------|
| <i>entity</i> | must be an Entity* |
|---------------|--------------------|

## Returns

void

Implements [Entity](#).

Definition at line 10 of file [Vehicle.cpp](#).

```
00010 {
00011     RoundStats::damageDone += getDamage\(\);
00012     entity->takeDamage(getDamage\(\));
00013 }
```

### 4.30.3.3 takeDamage()

```
void Vehicle::takeDamage (
    int damage ) [virtual]
```

Removes health from the vehicle object.

Preconditions:

- damage must be an int

Postconditions:

- Reduces the health of the vehicle object

## Parameters

|               |                |
|---------------|----------------|
| <i>damage</i> | must be an int |
|---------------|----------------|

## Returns

void

Implements [Entity](#).

Definition at line 6 of file [Vehicle.cpp](#).

```
00006 {
00007     setHealth(getHealth\(\) - damage);
00008 }
```

The documentation for this class was generated from the following files:

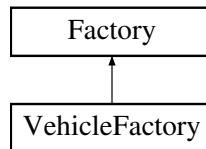
- Vehicle.h
- Vehicle.cpp

## 4.31 VehicleFactory Class Reference

[VehicleFactory](#) class.

```
#include <VehicleFactory.h>
```

Inheritance diagram for [VehicleFactory](#):



### Public Member Functions

- [VehicleFactory](#) ([Type](#) \*type, [AddOn](#) \*addOn)  
*Instantiates the vehicle factory.*
- [Entity](#) \* [createEntity](#) ([Alliance](#) \*alliance)  
*Instantiates and returns a vehicle for the given alliance.*
- [Factory](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current vehicle factory.*

#### 4.31.1 Detailed Description

[VehicleFactory](#) class.

Used to instantiate [Vehicle](#) objects.

Definition at line 10 of file [VehicleFactory.h](#).

#### 4.31.2 Constructor & Destructor Documentation

##### 4.31.2.1 VehicleFactory()

```
VehicleFactory::VehicleFactory (
    Type * type,
    AddOn * addOn )
```

Instantiates the vehicle factory.

Parameters

|              |                                   |
|--------------|-----------------------------------|
| <i>type</i>  | must be a <a href="#">Type</a> *  |
| <i>addOn</i> | must be a <a href="#">AddOn</a> * |

Definition at line 4 of file [VehicleFactory.cpp](#).

```
00004 : Factory(type, addOn) {}
```

### 4.31.3 Member Function Documentation

#### 4.31.3.1 clone()

```
Factory * VehicleFactory::clone ( ) [virtual]
```

Instantiates and returns a clone of the current vehicle factory.

Postconditions:

- Returns the clone of the current vehicle factory

#### Returns

Factory\* The vehicle factory clone

Implements [Factory](#).

Definition at line 18 of file [VehicleFactory.cpp](#).

```
00018 {
00019     return new VehicleFactory(getType()->clone(), getAddOn()->clone());
00020 }
```

#### 4.31.3.2 createEntity()

```
Entity * VehicleFactory::createEntity (
    Alliance * alliance ) [virtual]
```

Instantiates and returns a vehicle for the given alliance.

Preconditions:

- alliance must be an Alliance\*

Postconditions:

- Returns the instantiated vehicle object with specific state

#### Parameters

|                 |                     |
|-----------------|---------------------|
| <i>alliance</i> | must be a Alliance* |
|-----------------|---------------------|

**Returns**

Vehicle\* The instantiated vehicle

Implements [Factory](#).

Definition at line 6 of file [VehicleFactory.cpp](#).

```

00006         {
00007     Vehicle* v = new Vehicle(getType()->clone());
00008     v->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(v);
00012         return personnelAddOn;
00013     } else {
00014         return v;
00015     }
00016 }
```

The documentation for this class was generated from the following files:

- [VehicleFactory.h](#)
- [VehicleFactory.cpp](#)

## 4.32 WarEngine Class Reference

```
#include <WarEngine.h>
```

### Public Member Functions

- [WarEngineMemento](#) \* [saveState](#) ()  
*Captures current state of simulation via member variables and creates [WarEngineMemento](#) instance storing all relevant members in [WarEngineState](#).*
- void [loadSave](#) ([WarEngineMemento](#) \*save)  
*Takes in an instance of saved [WarEngine](#) states and sets current instance's member variables to memento state.*
- void [setWarTheatre](#) ([WarTheatre](#) \*battleGround)  
*Sets the state's area to passed in battleGround parameter.*
- void [addAlliance](#) ([Alliance](#) \*alliance)
- void [simulate](#) ()  
*Simulates the war.*

### Static Public Member Functions

- static [WarEngine](#) & [getInstance](#) ()  
*Function that returns a reference to the current (and only) instance of the class.*

### Protected Member Functions

- [WarEngine](#) ()  
*Constructor for class. Is responsible for ensuring only a single instance of class exists.*
- [WarEngine](#) (const [WarEngine](#) &)  
*Parameterized constructor for class.*
- [WarEngine](#) & [operator=](#) (const [WarEngine](#) &)  
*Overloaded operator = for class.*
- [~WarEngine](#) ()  
*Destructor for class responsible for freeing all allocated memory.*

### 4.32.1 Detailed Description

Class that contains all information regarding current simulation. Only one instance of class is allowed.

Definition at line 12 of file [WarEngine.h](#).

### 4.32.2 Constructor & Destructor Documentation

#### 4.32.2.1 WarEngine() [1/2]

```
WarEngine::WarEngine ( ) [protected]
```

Constructor for class. Is responsible for ensuring only a single instance of class exists.

Definition at line 5 of file [WarEngine.cpp](#).

```
00005     {
00006     this->state = new WarEngineState();
00007 }
```

#### 4.32.2.2 WarEngine() [2/2]

```
WarEngine::WarEngine (
    const WarEngine & ) [inline], [protected]
```

Parameterized constructor for class.

##### Parameters

|                       |                                   |
|-----------------------|-----------------------------------|
| <i>warEngine&amp;</i> | An anonymous warEngine reference. |
|-----------------------|-----------------------------------|

Postconditions:

- parameter must be of type [WarEngine&](#)

Definition at line 33 of file [WarEngine.h](#).

```
00033 {};
```

#### 4.32.2.3 ~WarEngine()

```
WarEngine::~WarEngine ( ) [protected]
```

Destructor for class responsible for freeing all allocated memory.

Definition at line 23 of file [WarEngine.cpp](#).

```
00023     {
00024     delete this->state;
00025 }
```

### 4.32.3 Member Function Documentation

#### 4.32.3.1 addAlliance()

```
void WarEngine::addAlliance (
    Alliance * alliance )
```

Definition at line 57 of file [WarEngine.cpp](#).

```
00057 {
00058     state->alliances.push_back(alliance);
00059 }
```

#### 4.32.3.2 getInstance()

```
WarEngine & WarEngine::getInstance ( ) [static]
```

Function that returns a reference to the current (and only) instance of the class.

##### Returns

[WarEngine&](#)

Definition at line 18 of file [WarEngine.cpp](#).

```
00018 {
00019     static WarEngine uniqueInstance_;
00020     return uniqueInstance_;
00021 }
```

#### 4.32.3.3 loadSave()

```
void WarEngine::loadSave (
    WarEngineMemento * save )
```

Takes in an instance of saved [WarEngine](#) states and sets current instance's member variables to memento state.

##### Preconditions:

- Save must be of type [WarEngineMemento\\*](#)

##### Postconditions:

- Sets the instance of the class's state member variable to the passed in save parameter.

## Parameters

|      |                             |
|------|-----------------------------|
| save | must be a WarEngineMemento* |
|------|-----------------------------|

## Returns

void

Definition at line 13 of file [WarEngine.cpp](#).

```

00013                                     {
00014     delete this->state;
00015     this->state = save->getState();
00016 }
```

## 4.32.3.4 operator=()

```

WarEngine & WarEngine::operator= (
    const WarEngine & ) [inline], [protected]
```

Overloaded operator = for class.

Definition at line 38 of file [WarEngine.h](#).

```

00038 { return *this; };
```

## 4.32.3.5 saveState()

```

WarEngineMemento * WarEngine::saveState ( )
```

Captures current state of simulation via member variables and creates [WarEngineMemento](#) instance storing all relevant members in [WarEngineState](#).

## Returns

WarEngineMemento\*

Definition at line 9 of file [WarEngine.cpp](#).

```

00009                                     {
00010     return new WarEngineMemento(state->clone());
00011 }
```

## 4.32.3.6 setWarTheatre()

```

void WarEngine::setWarTheatre (
    WarTheatre * battleGround )
```

Sets the state's area to passed in battleGround parameter.

## Parameters

|                     |                       |
|---------------------|-----------------------|
| <i>battleGround</i> | must be a WarTheatre* |
|---------------------|-----------------------|

## Preconditions:

- battleGround must be of type WarTheatre\*

## Postconditions:

- sets area in [WarEngineState](#) to passed in [WarTheatre](#).

## Returns

void

Definition at line 53 of file [WarEngine.cpp](#).

```
00053                                     {
00054         state->setArea(battleGround);
00055     }
```

## 4.32.3.7 simulate()

```
void WarEngine::simulate ( )
```

Simulates the war.

## PostConditions:

- Will simulate the war by running the engine

## Returns

void

Definition at line 28 of file [WarEngine.cpp](#).

```
00028     {
00029         vector<Alliance*> alliances = this->state->getAlliances();
00030
00031         for(int i = 0; i < alliances.size(); i++) {
00032             if (alliances[i]->getActive() == 1) {
00033                 RoundStats::clearStats();
00034                 state->getArea()->simulateBattle(alliances[i]);
00035                 cout << "===== " << endl;
00036                 cout << "Alliance " << alliances[i]->getID() << ":" << endl;
00037
00038                 if (alliances[i]->getID() == 2) {
00039                     cout << "Status: Surrendered" << endl;
00040                 } else if (alliances[i]->getID() == 3) {
00041                     cout << "Status: Found Peace" << endl;
00042                 } else {
00043                     cout << "Status: Active" << endl;
00044                 }
00045
00046                 cout << RoundStats::toString() << endl;
00047                 cout << "===== " << endl;
00048             }
00049         }
00050
00051     }
```

The documentation for this class was generated from the following files:

- WarEngine.h
- WarEngine.cpp



## 4.33 WarEngineMemento Class Reference

```
#include <WarEngineMemento.h>
```

### Friends

- class [WarEngine](#)

#### 4.33.1 Detailed Description

Class that encapsulates and externalises [WarEngine](#) State.

Definition at line 15 of file [WarEngineMemento.h](#).

#### 4.33.2 Friends And Related Function Documentation

##### 4.33.2.1 WarEngine

```
friend class WarEngine [friend]
```

Definition at line 17 of file [WarEngineMemento.h](#).

The documentation for this class was generated from the following files:

- WarEngineMemento.h
- WarEngineMemento.cpp

## 4.34 WarEngineState Class Reference

Class for storing current state of entire simulation.

```
#include <WarEngineState.h>
```

### Friends

- class [WarEngine](#)

#### 4.34.1 Detailed Description

Class for storing current state of entire simulation.

Class contains member variables areas which stores a vector of all war theatres and keypoints as well as a vector of all alliances in current simulation.

Definition at line 17 of file [WarEngineState.h](#).

## 4.34.2 Friends And Related Function Documentation

### 4.34.2.1 WarEngine

```
friend class WarEngine [friend]
```

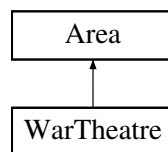
Definition at line 19 of file [WarEngineState.h](#).

The documentation for this class was generated from the following files:

- WarEngineState.h
- WarEngineState.cpp

## 4.35 WarTheatre Class Reference

Inheritance diagram for WarTheatre:



### Public Member Functions

- [WarTheatre](#) (std::string areaName)  
*Instantiates the war theatre.*
- [~WarTheatre](#) ()  
*Destroys the war theatre object.*
- bool [isKeyPoint](#) ()  
*Returns area type.*
- void [simulateBattle](#) ([Alliance](#) \*alliance)  
*Simulate Battle with troops from the alliance passed in.*
- void [addArea](#) ([Area](#) \*area)  
*Adds an area to the war theatre object.*
- void [addGeneral](#) ([General](#) \*general)  
*Adds a general to all the points held by the [WarTheatre](#).*
- [WarTheatre](#) \* [clone](#) ()  
*Instantiates and returns a clone of the current war theatre.*

### 4.35.1 Detailed Description

Definition at line 10 of file [WarTheatre.h](#).

## 4.35.2 Constructor & Destructor Documentation

### 4.35.2.1 WarTheatre()

```
WarTheatre::WarTheatre (
    std::string areaName )
```

Instantiates the war theatre.

Definition at line 5 of file [WarTheatre.cpp](#).

```
00005 :   Area(areaName) {}
```

### 4.35.2.2 ~WarTheatre()

```
WarTheatre::~~WarTheatre ( )
```

Destroys the war theatre object.

Postconditions:

- All dynamic memory should be deallocated from the war theatre object

Definition at line 7 of file [WarTheatre.cpp](#).

```
00007         {
00008     for (int i = 0; i < areas.size(); i++)
00009         delete areas[i];
00010 }
```

## 4.35.3 Member Function Documentation

### 4.35.3.1 addArea()

```
void WarTheatre::addArea (
    Area * area )
```

Adds an area to the war theatre object.

Preconditions:

- area must be an Area\*

Postconditions:

- Add area to war theatre object

**Parameters**

|             |                  |
|-------------|------------------|
| <i>area</i> | must be an Area* |
|-------------|------------------|

**Returns**

void

Definition at line 21 of file [WarTheatre.cpp](#).

```
00021 {  
00022     areas.push_back(area);  
00023 }
```

**4.35.3.2 addGeneral()**

```
void WarTheatre::addGeneral (  
    General * general ) [virtual]
```

Adds a general to all the points held by the [WarTheatre](#).

Precoditions:

- general must be a General\*

Postconditions:

- Add general to all points

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>general</i> | must be a General* |
|----------------|--------------------|

Implements [Area](#).

Definition at line 34 of file [WarTheatre.cpp](#).

```
00034 {  
00035     for (int i = 0; i < areas.size(); i++)  
00036         areas[i]->addGeneral(general);  
00037 }
```

**4.35.3.3 clone()**

```
WarTheatre * WarTheatre::clone ( ) [virtual]
```

Instantiates and returns a clone of the current war theatre.

Postconditions:

- Returns the clone of the current war theatre

**Returns**

WarTheatre\* The war theatre clone

Implements [Area](#).

Definition at line 25 of file [WarTheatre.cpp](#).

```
00025     {
00026         WarTheatre* w = new WarTheatre(getAreaName());
00027
00028         for (int i = 0; i < areas.size(); i++)
00029             w->addArea(areas[i]->clone());
00030
00031         return w;
00032     }
```

**4.35.3.4 isKeyPoint()**

```
bool WarTheatre::isKeyPoint ( ) [virtual]
```

Returns area type.

Postconditions:

- Returns false

**Returns**

bool The area type

Implements [Area](#).

Definition at line 12 of file [WarTheatre.cpp](#).

```
00012     {
00013         return false;
00014     }
```

**4.35.3.5 simulateBattle()**

```
void WarTheatre::simulateBattle (
    Alliance * alliance ) [virtual]
```

Simulate Battle with troops from the alliance passed in.

Preconditions:

- alliance must be an Alliance\*

Postconditions:

- Call attacks function of areas

## Parameters

|                 |                      |
|-----------------|----------------------|
| <i>alliance</i> | must be an Alliance* |
|-----------------|----------------------|

## Returns

void

Implements [Area](#).Definition at line 16 of file [WarTheatre.cpp](#).

```

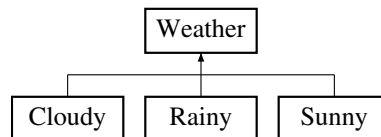
00016                                     {
00017     for (int i = 0; i < areas.size(); i++)
00018         areas[i]->simulateBattle(alliance);
00019 }
```

The documentation for this class was generated from the following files:

- WarTheatre.h
- WarTheatre.cpp

## 4.36 Weather Class Reference

Inheritance diagram for Weather:



### Public Member Functions

- [Weather](#) ()  
*Instantiates the [Weather](#) object.*
- [~Weather](#) ()  
*Destructor for the [Weather](#) object.*
- double [getMultiplier](#) ()  
*Returns double which shows the weather multiplier.*
- virtual void [handleChange](#) ([KeyPoint](#) \*k)=0
- virtual std::string [getWeather](#) ()=0
- virtual [Weather](#) \* [clone](#) ()=0

### Protected Attributes

- double [multiplier](#)

### 4.36.1 Detailed Description

Definition at line 9 of file [Weather.h](#).

### 4.36.2 Constructor & Destructor Documentation

#### 4.36.2.1 Weather()

```
Weather::Weather ( )
```

Instantiates the [Weather](#) object.

Definition at line 3 of file [Weather.cpp](#).  
00003 { }

#### 4.36.2.2 ~Weather()

```
Weather::~Weather ( )
```

Destructor for the [Weather](#) object.

Definition at line 5 of file [Weather.cpp](#).  
00005 { }

### 4.36.3 Member Function Documentation

#### 4.36.3.1 clone()

```
virtual Weather * Weather::clone ( ) [pure virtual]
```

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

#### 4.36.3.2 getMultiplier()

```
double Weather::getMultiplier ( )
```

Returns double which shows the weather multiplier.

Postconditions:

- Returns the double multiplier

##### Returns

double which is the multiplier

Definition at line 7 of file [Weather.cpp](#).

```
00007 {  
00008     return this->multiplier;  
00009 }
```

#### 4.36.3.3 getWeather()

```
virtual std::string Weather::getWeather ( ) [pure virtual]
```

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

#### 4.36.3.4 handleChange()

```
virtual void Weather::handleChange (  
    KeyPoint * k ) [pure virtual]
```

Implemented in [Cloudy](#), [Rainy](#), and [Sunny](#).

### 4.36.4 Member Data Documentation

#### 4.36.4.1 multiplier

```
double Weather::multiplier [protected]
```

Definition at line 12 of file [Weather.h](#).

The documentation for this class was generated from the following files:

- [Weather.h](#)
- [Weather.cpp](#)



## Chapter 5

# File Documentation

### 5.1 AddOn.cpp

```
00001 #include "AddOn.h"
00002
00003 AddOn::AddOn(int value): Entity() {
00004     this->value = value;
00005     entity = NULL;
00006 }
00007
00008 void AddOn::setValue(int value) {
00009     this->value = value;
00010 }
00011
00012 int AddOn::getValue() {
00013     return value;
00014 }
00015
00016 void AddOn::setEntity(Entity* entity) {
00017     this->entity = entity;
00018 }
00019
00020 Entity* AddOn::getEntity() {
00021     return this->entity;
00022 }
```

### 5.2 AddOn.h

```
00001 #ifndef ADDON_H
00002 #define ADDON_H
00003 #include "Entity.h"
00004
00010 class AddOn : public Entity {
00011
00012 protected:
00013     int value;
00014     Entity* entity;
00015
00016 public:
00022     AddOn(int value);
00023
00036     void setValue(int value);
00037
00046     int getValue();
00047
00060     void setEntity(Entity* entity);
00061
00070     Entity* getEntity();
00071
00072     virtual void takeDamage(int damage) = 0;
00073
00074     virtual void dealDamage(Entity* entity) = 0;
00075
00076     virtual AddOn* clone() = 0;
00077 };
00078
00079 #endif
```

## 5.3 AddOnTest.h

```

00001 #include <stdexcept>
00002 #include "AddOn.h"
00003 #include "Piercing.h"
00004 #include "Armour.h"
00005 #include "Personnel.h"
00006 #include "TerrainType.h"
00007 #include "gtest/gtest.h"
00008
00009 namespace {
00010
00011     // Tests General AddOn Functionality
00012
00013     // Tests AddOn setValue()
00014     // ===== Precondition Testing =====
00015     // Test Precondition Negative
00016     TEST(AddOnSetValueTest, TestPreconditionNegative) {
00017         Armour* a = new Armour(10);
00018         try {
00019             a->setValue(-5);
00020             FAIL();
00021         } catch (std::invalid_argument& err) {
00022             EXPECT_EQ(err.what(), std::string("value must be greater than zero"));
00023         } catch (...) {
00024             FAIL();
00025         }
00026     }
00027
00028     // Test Precondition Positive
00029     TEST(AddOnSetValueTest, TestPreconditionPositive) {
00030         Armour* a = new Armour(5);
00031         a->setValue(5);
00032         EXPECT_EQ(5, a->getValue());
00033     }
00034
00035     // Test Precondition Bounds
00036     TEST(AddOnSetValueTest, TestPreconditionBounds) {
00037         Armour* a = new Armour(5);
00038         try {
00039             a->setValue(0);
00040             FAIL();
00041         } catch (std::invalid_argument& err) {
00042             EXPECT_EQ(err.what(), std::string("value must be greater than zero"));
00043         } catch (...) {
00044             FAIL();
00045         }
00046     }
00047
00048     // ===== Positive Testing =====
00049     // Test Preconditions Bounds
00050     TEST(AddOnSetValueTest, PositiveTesting) {
00051         Armour* a = new Armour(5);
00052
00053         a->setValue(5);
00054         EXPECT_EQ(5, a->getValue());
00055
00056         a->setValue(10);
00057         EXPECT_EQ(10, a->getValue());
00058
00059         a->setValue(20);
00060         EXPECT_EQ(20, a->getValue());
00061
00062         a->setValue(55);
00063         EXPECT_EQ(55, a->getValue());
00064
00065         a->setValue(3);
00066         EXPECT_EQ(3, a->getValue());
00067
00068         a->setValue(100);
00069         EXPECT_EQ(100, a->getValue());
00070     }
00071
00072     // Tests AddOn setEntity()
00073     // ===== Positive Testing =====
00074     // Test Preconditions Bounds
00075     TEST(AddOnSetEntityTest, PositiveTesting) {
00076         Armour* a = new Armour(5);
00077
00078         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00079         a->setEntity(p);
00080         EXPECT_EQ(p, a->getEntity());
00081
00082         Personnel* m = new Personnel(new TerrainType(), 100, 10);
00083         a->setEntity(m);
00084         EXPECT_EQ(m, a->getEntity());
00085     }

```

```

00086         Personnel* n = new Personnel(new TerrainType(), 100, 10);
00087         a->setEntity(n);
00088         EXPECT_EQ(n, a->getEntity());
00089     }
00090
00091     // Tests Armour AddOn Functionality
00092
00093     // Tests Armour takeDamage()
00094     // ===== Precondition Testing =====
00095     // Test Precondition Negative
00096     TEST(ArmourTakeDamageTest, TestPreconditionNegative) {
00097         Armour* a = new Armour(10);
00098         try {
00099             a->takeDamage(-5);
00100             FAIL();
00101         } catch (std::invalid_argument& err) {
00102             EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
00103         } catch (...) {
00104             FAIL();
00105         }
00106     }
00107
00108     // Test Precondition Positive
00109     TEST(ArmourTakeDamageTest, TestPreconditionPositive) {
00110         Armour* a = new Armour(10);
00111         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00112
00113         a->setEntity(p);
00114         a->takeDamage(10);
00115         EXPECT_EQ(0, a->getValue());
00116         EXPECT_EQ(100, p->getHealth());
00117     }
00118
00119     // Test Precondition Bounds
00120     TEST(ArmourTakeDamageTest, TestPreconditionBounds) {
00121         Armour* a = new Armour(10);
00122         try {
00123             a->takeDamage(0);
00124             FAIL();
00125         } catch (std::invalid_argument& err) {
00126             EXPECT_EQ(err.what(), std::string("damage must be greater than zero"));
00127         } catch (...) {
00128             FAIL();
00129         }
00130     }
00131
00132     // ===== Positive Testing =====
00133     // Test Preconditions Bounds
00134     TEST(ArmourTakeDamageTest, PositiveTesting) {
00135         Armour* a = new Armour(20);
00136         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00137         a->setEntity(p);
00138
00139         a->takeDamage(10);
00140         EXPECT_EQ(10, a->getValue());
00141         EXPECT_EQ(100, p->getHealth());
00142
00143         a->takeDamage(10);
00144         EXPECT_EQ(0, a->getValue());
00145         EXPECT_EQ(100, p->getHealth());
00146
00147         a->takeDamage(10);
00148         EXPECT_EQ(0, a->getValue());
00149         EXPECT_EQ(90, p->getHealth());
00150     }
00151
00152     // Tests Armour dealDamage()
00153     // ===== Positive Testing =====
00154     // Test Preconditions Bounds
00155     TEST(ArmourDealDamageTest, PositiveTesting) {
00156         Armour* a = new Armour(10);
00157         Personnel* p = new Personnel(new TerrainType(), 100, 10);
00158         a->setEntity(p);
00159         Personnel* x = new Personnel(new TerrainType(), 100, 10);
00160
00161         a->dealDamage(x);
00162         EXPECT_EQ(10, a->getValue());
00163         EXPECT_EQ(90, x->getHealth());
00164
00165         a->dealDamage(x);
00166         EXPECT_EQ(10, a->getValue());
00167         EXPECT_EQ(80, x->getHealth());
00168
00169         a->dealDamage(x);
00170         EXPECT_EQ(10, a->getValue());
00171         EXPECT_EQ(70, x->getHealth());
00172     }

```

```

00173
00174 // Tests Piercing AddOn Functionality
00175
00176 // Tests Piercing takeDamage()
00177 // ===== Positive Testing =====
00178 // Test Preconditions Bounds
00179 TEST(PiercingTakeDamageTest, PositiveTesting) {
00180     Piercing* pi = new Piercing(10);
00181     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00182     pi->setEntity(p);
00183
00184     pi->takeDamage(10);
00185     EXPECT_EQ(10, pi->getValue());
00186     EXPECT_EQ(90, p->getHealth());
00187
00188     pi->takeDamage(10);
00189     EXPECT_EQ(10, pi->getValue());
00190     EXPECT_EQ(80, p->getHealth());
00191
00192     pi->takeDamage(10);
00193     EXPECT_EQ(10, pi->getValue());
00194     EXPECT_EQ(70, p->getHealth());
00195 }
00196
00197 // Tests Piercing dealDamage()
00198 // ===== Positive Testing =====
00199 // Test Preconditions Bounds
00200 TEST(PiercingDealDamageTest, PositiveTesting) {
00201     Piercing* pi = new Piercing(10);
00202     Personnel* p = new Personnel(new TerrainType(), 100, 10);
00203     pi->setEntity(p);
00204     Personnel* x = new Personnel(new TerrainType(), 100, 10);
00205
00206     pi->dealDamage(x);
00207     EXPECT_EQ(10, pi->getValue());
00208     EXPECT_EQ(80, x->getHealth());
00209
00210     pi->dealDamage(x);
00211     EXPECT_EQ(10, pi->getValue());
00212     EXPECT_EQ(60, x->getHealth());
00213
00214     pi->dealDamage(x);
00215     EXPECT_EQ(10, pi->getValue());
00216     EXPECT_EQ(40, x->getHealth());
00217 }
00218 }

```

## 5.4 AerialType.cpp

```

00001 #include "AerialType.h"
00002
00003 AerialType::AerialType() {}
00004
00005 string AerialType::getTypeDesc() {
00006     return "Aerial";
00007 }
00008
00009 Type* AerialType::clone() {
00010     return new AerialType();
00011 }

```

## 5.5 AerialType.h

```

00001 #ifndef AERIALTYPE_H
00002 #define AERIALTYPE_H
00003
00004 #include "Type.h"
00005
00011 class AerialType : public Type {
00012
00013 public:
00017     AerialType();
00018
00027     string getTypeDesc();
00028
00029     Type* clone();
00030 };
00031
00032 #endif

```

## 5.6 Aggressive.cpp

```

00001 #include "Aggressive.h"
00002 #include "KeyPoint.h"
00003
00004 Aggressive::Aggressive() {}
00005
00006 void Aggressive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
00007     int randomNumber = (rand() % 10) + 5;
00008     keyPoint->moveEntitiesInto(alliance, randomNumber);
00009 }
00010
00011 Strategy* Aggressive::clone() {
00012     return new Aggressive();
00013 }

```

## 5.7 Aggressive.h

```

00001 #ifndef AGGRESSIVE_H
00002 #define AGGRESSIVE_H
00003 #include "Strategy.h"
00004
00005 class Aggressive : public Strategy {
00006
00007 public:
00008     Aggressive();
00009
00023     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00024
00030     Strategy* clone();
00031 };
00032
00033 #endif

```

## 5.8 Alliance.cpp

```

00001 #include "Alliance.h"
00002 #include "Negotiator.h"
00003 #include "Entity.h"
00004 #include "RoundStats.h"
00005 #include <time.h>
00006 #include <iostream>
00007
00008 using namespace std;
00009
00010 int Alliance::totalNum = 0;
00011
00012 Alliance::Alliance() {
00013     this->active = 1;
00014     this->aID = totalNum++;
00015     this->negotiator = NULL;
00016     srand(time(0));
00017 }
00018
00019 Alliance::Alliance(Alliance& alliance) {
00020     this->active = alliance.active;
00021     this->aID = alliance.aID;
00022
00023     for (int i = 0; i < alliance.members.size(); i++)
00024         this->addCountry(alliance.members[i]->clone());
00025
00026     for (int i = 0; i < alliance.production.size(); i++)
00027         this->addFactory(alliance.production[i]->clone());
00028
00029     for (int i = 0; i < alliance.reserveEntities.size(); i++)
00030         this->addReserveEntity(alliance.reserveEntities[i]->clone());
00031
00032     this->negotiator = NULL;
00033 }
00034
00035 Alliance::~Alliance() {
00036
00037     for (int i = 0; i < members.size(); i++)
00038         //delete members[i];
00039
00040     if (this->negotiator != NULL) {
00041         this->negotiator->removeAlliance(this);
00042
00043         if (this->negotiator->getNumAlliances() == 1)

```

```

00044         delete this->negotiator;
00045     }
00046 }
00047
00048 void Alliance::setNegotiator(Negotiator* negotiator) {
00049     this->negotiator = negotiator;
00050 }
00051
00052 void Alliance::addCountry(Country* nation) {
00053     members.push_back(nation);
00054 }
00055
00056 vector<Entity*> Alliance::getReserveEntities(int number) {
00057     vector<Entity*> out;
00058     for (int i = 0; i < number && i < reserveEntities.size(); i++) {
00059         out.push_back(reserveEntities[i]);
00060         reserveEntities.erase(reserveEntities.begin() + i);
00061     }
00062
00063     return out;
00064 }
00065
00066 void Alliance::addReserveEntity(Entity* entity) {
00067     reserveEntities.push_back(entity);
00068 }
00069
00070 int Alliance::numRemainingEntities() {
00071     return reserveEntities.size();
00072 }
00073
00074 bool Alliance::considerPeace() {
00075     return (rand() % 2 == 0);
00076 }
00077
00078 void Alliance::addFactory(Factory* factory) {
00079     production.push_back(factory);
00080 }
00081
00082 void Alliance::runFactories() {
00083     for (int i = 0; i < production.size(); i++) {
00084         RoundStats::numEntitiesCreated++;
00085         reserveEntities.push_back(production[i]->createEntity(this));
00086     }
00087 }
00088
00089 void Alliance::surrender() {
00090     this->active = 2; //Number 2 means that Alliance has surrendered
00091
00092     this->negotiator->removeAlliance(this);
00093 }
00094
00095 int Alliance::getID() {
00096     return this->aID;
00097 }
00098
00099 bool Alliance::offerPeace() {
00100
00101     if (this->negotiator->sendPeace(this)) //Send the peace deal to all the alliances fighting against
00102     {
00103         this->active = 3; //Number 3 means that Alliance chose to peacefully pull out of war
00104         return true;
00105     }
00106
00107     return false;
00108 }
00109
00110 int Alliance::getActive() {
00111     return active;
00112 }
00113
00114 Alliance* Alliance::clone() {
00115     return new Alliance(*this);
00116 }

```

## 5.9 Alliance.h

```

00001 #ifndef ALLIANCE_H
00002 #define ALLIANCE_H
00003 #include "Country.h"
00004 #include "Factory.h"
00005 #include "Country.h"
00006 #include <vector>
00007

```

```

00008 class Negotiator;
00009 class Entity;
00010
00011 using namespace std;
00012
00013 class Alliance {
00014 private:
00015     static int totalNum;
00016     int aID;
00017     vector<Factory*> production;
00018     Negotiator* negotiator;
00019     vector<Country*> members;
00020     int active;
00021     vector<Entity*> reserveEntities;
00022
00023 public:
00024     Alliance();
00025
00026     Alliance(Alliance& alliance);
00027
00028     ~Alliance();
00029
00030     void setNegotiator(Negotiator* newNegotiator);
00031
00032     void addCountry(Country* nation);
00033
00034     vector<Entity*> getReserveEntities(int number);
00035
00036     void addReserveEntity(Entity* entity);
00037
00038     int numRemainingEntities();
00039
00040     bool considerPeace();
00041
00042     void addFactory(Factory* factory);
00043
00044     void runFactories();
00045
00046     void surrender();
00047
00048     int getID();
00049
00050     bool offerPeace();
00051
00052     Alliance* clone();
00053
00054     void setActiveStatus(bool active);
00055
00056     int getActive();
00057 };
00058 #endif

```

## 5.10 AquaticType.cpp

```

00001 #include "AquaticType.h"
00002
00003 using namespace std;
00004
00005 AquaticType::AquaticType() {}
00006
00007 string AquaticType::getTypeDesc() {
00008     return "Aquatic";
00009 }
00010
00011 Type* AquaticType::clone() {
00012     return new AquaticType();
00013 }

```

## 5.11 AquaticType.h

```

00001 #ifndef AQUATICTYPE_H
00002 #define AQUATICTYPE_H
00003
00004 #include "Type.h"
00005
00006 class AquaticType : public Type {
00007

```

```
00013 public:
00017     AquaticType();
00018
00027     string getTypeDesc();
00028
00029     Type* clone();
00030 };
00031
00032 #endif
```

## 5.12 Area.cpp

```
00001 #include "Area.h"
00002
00003 using namespace std;
00004
00005 Area::Area(string areaName) {
00006     this->areaName = areaName;
00007 }
00008
00009 Area::~Area() {}
00010
00011 std::string Area::getAreaName() const {
00012     return areaName;
00013 }
```

## 5.13 Area.h

```
00001 #ifndef AREA_H
00002 #define AREA_H
00003 #include <string>
00004 #include "Alliance.h"
00005
00006 class General;
00007
00008 class Area {
00009
00010 private:
00011     std::string areaName;
00012
00013 public:
00017     Area(std::string areaName);
00018
00022     virtual ~Area();
00023
00024     virtual bool isKeyPoint() = 0;
00025
00026     virtual void simulateBattle(Alliance* alliance) = 0;
00027
00033     std::string getAreaName() const;
00034
00035     virtual Area* clone() = 0;
00036
00037     virtual void addGeneral(General* general) = 0;
00038 };
00039
00040 #endif
```

## 5.14 Armour.cpp

```
00001 #include "Armour.h"
00002
00003 Armour::Armour(int value) : AddOn(value) {}
00004
00005 void Armour::takeDamage(int damage) {
00006     if (value > 0) {
00007         value -= damage;
00008     } else {
00009         entity->takeDamage(damage);
00010     }
00011 }
00012
00013 void Armour::dealDamage(Entity* entity) {
00014     this->entity->dealDamage(entity);
00015 }
```



```

00016
00017 AddOn* Armour::clone() {
00018     Armour* armour = new Armour(value);
00019     if (getEntity() != NULL)
00020         armour->setEntity(entity->clone());
00021     return armour;
00022 }

```

## 5.15 Armour.h

```

00001 #ifndef ARMOUR_H
00002 #define ARMOUR_H
00003 #include "AddOn.h"
00004 #include "Entity.h"
00005
00011 class Armour : public AddOn {
00012
00013
00014 public:
00020     Armour(int value);
00021
00034     void takeDamage(int damage);
00035
00048     void dealDamage(Entity* entity);
00049
00058     AddOn* clone();
00059 };
00060
00061 #endif

```

## 5.16 Cloudy.cpp

```

00001 #include "Cloudy.h"
00002 #include "Rainy.h"
00003
00004 Cloudy::Cloudy(): Weather() {
00005     this->multiplier = 0.75;
00006 }
00007
00008 std::string Cloudy::getWeather() {
00009     return "Cloudy";
00010 }
00011
00012 void Cloudy::handleChange(KeyPoint* k) {
00013     Rainy* newWeather = new Rainy();
00014     k->setWeather(newWeather);
00015 }
00016
00017 Weather* Cloudy::clone() {
00018     return new Cloudy();
00019 }

```

## 5.17 Cloudy.h

```

00001 #ifndef CLOUDY_H
00002 #define CLOUDY_H
00003 #include "Weather.h"
00004 #include <string>
00005
00006 class Cloudy : public Weather {
00007
00008 public:
00012     Cloudy();
00013
00022     std::string getWeather();
00023
00036     void handleChange(KeyPoint* k);
00037
00043     Weather* clone();
00044 };
00045
00046 #endif

```

## 5.18 Country.cpp

```

00001 #include "Country.h"
00002
00003 using namespace std;
00004
00005 Country::Country(std::string name){
00006     this->name = name;
00007     this->id = rand() % 1000;
00008 }
00009
00010
00011 Country* Country::clone(){
00012     return new Country(this->name);
00013 }
00014
00015 string Country::getName() const{
00016     return this->name;
00017 }
00018
00019 int Country::getID() const{
00020     return this->id;
00021 }
00022
00023

```

## 5.19 Country.h

```

00001 #ifndef COUNTRY_H
00002 #define COUNTRY_H
00003 #include <string>
00004
00005 class Country {
00006
00007 private:
00008     std::string name;
00009     int id;
00010
00011 public:
00012     Country(std::string name);
00013
00014     Country* clone();
00015
00016     std::string getName() const;
00017
00018     int getID() const;
00019 };
00020
00021 #endif

```

## 5.20 Defensive.cpp

```

00001 #include "Defensive.h"
00002
00003 Defensive::Defensive() {
00004
00005 }
00006
00007 void Defensive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
00008
00009     int randomNumber = (rand() % 5) + 1;
00010     keyPoint->moveEntitiesInto(alliance, randomNumber);
00011 }
00012
00013 Strategy* Defensive::clone() {
00014     return new Defensive();
00015 }

```

## 5.21 Defensive.h

```

00001 #ifndef DEFENSIVE_H
00002 #define DEFENSIVE_H
00003 #include "Strategy.h"
00004 #include "KeyPoint.h"

```

```

00005 #include "Alliance.h"
00006 #include "Personnel.h"
00007 class Defensive : public Strategy {
00008
00009
00010 public:
00011     Defensive();
00012
00022     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00023
00029     Strategy* clone();
00030 };
00031
00032 #endif

```

## 5.22 EasySetup.cpp

```

00001 #include "EasySetup.h"
00002 #include <string.h>
00003 #include "Alliance.h"
00004 #include "Country.h"
00005 #include "AquaticType.h"
00006 #include "AerialType.h"
00007 #include "TerrainType.h"
00008 #include "Piercing.h"
00009 #include "Armour.h"
00010 #include "PersonnelFactory.h"
00011 #include "VehicleFactory.h"
00012 #include "SupportFactory.h"
00013 #include "KeyPoint.h"
00014 #include "WarTheatre.h"
00015 #include "Passive.h"
00016 #include "Aggressive.h"
00017 #include "Defensive.h"
00018 #include "WarEngine.h"
00019 #include "Negotiator.h"
00020
00021 EasySetup::EasySetup() {
00022     saveArchive = new SaveArchive();
00023 }
00024
00025 void EasySetup::setupSimulation() {
00026     while (true)
00027     {
00028         cout << "Load simulation (L) or New Simulation (N): ";
00029         string selectedOption;
00030         cin >> selectedOption;
00031         cin.ignore();
00032
00033         if(toupper(selectedOption[0]) == 'L')
00034         {
00035             string saveName;
00036             cout << "Please enter the name of the save to be re-simulated" << endl;
00037             getline(cin, saveName); // getting the name of the save-archive
00038             try {
00039                 this->loadSpecificSave(saveName); // loading the save-archive
00040                 return; // will return if the above the function does not throw an exception
00041             } catch(const std::exception& exception) {
00042                 cout << "Error: " << exception.what() << endl;
00043
00044                 if (strcmp(exception.what(), "Save archive is empty") == 0) {
00045                     cout << "Please create new simulation" << endl;
00046                     goto setup;
00047
00048                 } else if (strcmp(exception.what(), "No save with given name exists") == 0) {
00049                     cout << "Please enter the correct name of save-archive and try again or create new
00050 simulation" << endl;
00051                 }
00052             }
00053
00054             } else if(toupper(selectedOption[0]) == 'N') {
00055                 // setting up a new simulation
00056                 goto setup;
00057             } else {
00058                 cout << "Incorrect input: Please enter (L) or (N)" << endl;
00059             }
00060         }
00061
00062         setup:
00063         // Creating alliances and generals
00064         int numAlliesAndGenerals;
00065         cout << "Enter number of alliances: ";

```

```

00066         cin >> numAlliesAndGenerals;
00067
00068         Alliance** alliances = new Alliance*[numAlliesAndGenerals];
00069         General** generals = new General*[numAlliesAndGenerals];
00070
00071         int numCountries,
00072             numFactories;
00073         string countryName,
00074             factoryType,
00075             selectedFactory,
00076             selectedAddOn;
00077         Country* country;
00078         Type* type;
00079         AddOn* addOn;
00080         Factory* factory;
00081
00082         Negotiator* negotiator = new Negotiator();
00083
00084         for (int i = 0; i < numAlliesAndGenerals; i++) {
00085             alliances[i] = new Alliance();
00086             negotiator->addAlliance(alliances[i]);
00087             alliances[i]->setNegotiator(negotiator);
00088             WarEngine::getInstance().addAlliance(alliances[i]);
00089
00090             cout << "Enter number of countries for Alliance " << alliances[i]->getID() << ": ";
00091             cin >> numCountries;
00092             cin.ignore();
00093
00094             for (int k = 0; k < numCountries; k++) {
00095                 cout << "Enter name of county " << k+1 << ": ";
00096                 getline(cin, countryName);
00097                 country = new Country(countryName);
00098                 alliances[i]->addCountry(country);
00099             }
00100
00101             cout << "Enter number of factories for Alliance " << alliances[i]->getID() << ": ";
00102             cin >> numFactories;
00103
00104             for (int k = 0; k < numFactories; k++) {
00105                 retryType:
00106                 cout << "Factory " << k+1 << " is of type Aquatic(Q), Aerial(E), or Terrain(T) : ";
00107                 cin >> factoryType;
00108                 cin.ignore();
00109
00110                 if (toupper(factoryType[0]) == 'Q') {
00111                     type = new AerialType;
00112                 } else if (toupper(factoryType[0]) == 'E') {
00113                     type = new AerialType;
00114                 } else if (toupper(factoryType[0]) == 'T') {
00115                     type = new TerrainType;
00116                 } else {
00117                     cout << "Invalid type input! Try again" << endl;
00118                     goto retryType;
00119                 }
00120
00121                 retryAddOn:
00122                 cout << "Select AddOn for factory " << k+1 << " Armour(A), Piercing(P) or None(N) : ";
00123                 getline(cin, selectedAddOn);
00124                 if (toupper(selectedAddOn[0]) == 'A') {
00125                     int value;
00126                     cout << "Enter armour value: ";
00127                     cin >> value;
00128                     cin.ignore();
00129                     addOn = new Armour(value);
00130                 } else if (toupper(selectedAddOn[0]) == 'P') {
00131                     int value;
00132                     cout << "Enter piercing value: ";
00133                     cin >> value;
00134                     cin.ignore();
00135                     addOn = new Piercing(value);
00136                 } else if (toupper(selectedAddOn[0]) == 'N') {
00137                     addOn = NULL;
00138                 } else {
00139                     cout << "Invalid AddOn input! Try again" << endl;
00140                     goto retryAddOn;
00141                 }
00142
00143                 retryFactory:
00144                 cout << "Which factory is factory " << k+1 << " Vehicle(V), Personnel(P), or Support(S) : ";
00145                 getline(cin, selectedFactory);
00146                 if (toupper(selectedFactory[0]) == 'V') {
00147                     factory = new VehicleFactory(type, addOn);
00148                 } else if (toupper(selectedFactory[0]) == 'P') {
00149                     factory = new PersonnelFactory(type, addOn);
00150                 } else if (toupper(selectedFactory[0]) == 'S') {
00151                     factory = new SupportFactory(type, addOn);

```

```

00152         } else {
00153             cout << "Invalid factory input! Try again" << endl;
00154             goto retryFactory;
00155         }
00156
00157         alliances[i]->addFactory(factory);
00158     }
00159
00160     string selectedStrat;
00161     Strategy* strat;
00162
00163     retryStrat:
00164     cout << "What is this Alliances generals strategy Passive(P), Defensive(D), or
Aggressive(A) : ";
00165     getline(cin, selectedStrat);
00166     if (toupper(selectedStrat[0]) == 'P') {
00167         strat = new Passive();
00168     } else if (toupper(selectedStrat[0]) == 'D') {
00169         strat = new Defensive();
00170     } else if (toupper(selectedStrat[0]) == 'A') {
00171         strat = new Aggressive();
00172     } else {
00173         cout << "Invalid strategy input! Try again" << endl;
00174         goto retryStrat;
00175     }
00176
00177     generals[i] = new General(alliances[i], strat);
00178 }
00179
00180 int factoryRun;
00181 cout << "How many production runs do you wish to perform: ";
00182 cin >> factoryRun;
00183 cin.ignore();
00184 for (int i = 0; i < numAlliesAndGenerals; i++) {
00185     for (int j = 0; j < factoryRun; j++) {
00186         alliances[i]->runFactories();
00187     }
00188 }
00189
00190 // Creating main WarTheatre
00191 WarTheatre* mainBattleGround;
00192 cout << "Creating the main battle ground" << endl;
00193 string battleGroundName;
00194 cout << "Set main battle ground's name: ";
00195 getline(cin, battleGroundName);
00196 mainBattleGround = new WarTheatre(battleGroundName);
00197
00198 int sizeOfGrounds;
00199 cout << "Enter number of battle grounds in " << battleGroundName << " battle ground: ";
00200 cin >> sizeOfGrounds;
00201 cin.ignore();
00202 WarTheatre** battleGrounds = new WarTheatre*[sizeOfGrounds];
00203
00204 // Creating sub WarTheatres
00205 for (int i = 0; i < sizeOfGrounds; i++) {
00206     battleGroundName.clear();
00207     cout << "Set battle ground " << i+1 << "'s name: ";
00208     getline(cin, battleGroundName);
00209     battleGrounds[i] = new WarTheatre(battleGroundName);
00210 }
00211
00212 vector<int> numKeyPoints;
00213 int numKeyPoint = 0;
00214
00215 for (int i = 0; i < sizeOfGrounds; i++) {
00216     cout << "Enter number of key points in " << battleGrounds[i]->getAreaName() << " battle
ground: ";
00217     cin >> numKeyPoint;
00218     cin.ignore();
00219     numKeyPoints.push_back(numKeyPoint);
00220     numKeyPoint = 0;
00221 }
00222
00223 KeyPoint* keyPoint;
00224 string keyPointName;
00225 int numEntitiesInKeyPt;
00226
00227 // Creating KeyPoints for the sub WarTheatres
00228 for (int i = 0; i < sizeOfGrounds; i++) {
00229     numKeyPoint = numKeyPoints[i];
00230     cout << "For " << battleGrounds[i]->getAreaName() << "'s key points" << endl;
00231
00232     for (int k = 0; k < numKeyPoint; k++) {
00233         cout << "Set key point " << i+1 << "'s name: ";
00234         getline(cin, keyPointName);
00235         keyPoint = new KeyPoint(keyPointName);
00236     }

```

```

00237         for (int a = 0; a < numAlliesAndGenerals; a++) {
00238             tryAgain:
00239                 cout << "There are " << alliances[a]->numRemainingEntities() << " for Alliance " <<
a+1 << endl;
00240                 cout << "How many would you like to place in " << keyPointName << " keypoint? ";
00241                 cin >> numEntitiesInKeyPt;
00242                 cin.ignore();
00243
00244                 if (alliances[a]->numRemainingEntities() > 0 &&
alliances[a]->numRemainingEntities() < numEntitiesInKeyPt) {
00245                     cout << "You selected more than the available amount. Try again " << endl;
00246                     goto tryAgain;
00247                 } else if (alliances[a]->numRemainingEntities() <= 0) {
00248                     continue;
00249                 } else {
00250                     keyPoint->moveEntitiesInto(alliances[a], numEntitiesInKeyPt);
00251                 }
00252             }
00253             battleGrounds[i]->addArea(keyPoint);
00254         }
00255         mainBattleGround->addArea(battleGrounds[i]);
00256     }
00257     for (int i = 0; i < numAlliesAndGenerals; i++) {
00258         mainBattleGround->addGeneral(generals[i]);
00259     }
00260     WarEngine::getInstance().setWarTheatre(mainBattleGround);
00261 }
00262 void EasySetup::runSimulation() {
00263     WarEngine::getInstance().simulate();
00264 }
00265 void EasySetup::saveSimulationSetup() {
00266     // Getting the name of the save
00267     cout << "Please enter name of save: ";
00268     string saveName;
00269     getline(cin, saveName);
00270
00271     // saving the current state of the simulation
00272     saveArchive->addNewSave(saveName, WarEngine::getInstance().saveState());
00273 }
00274 void EasySetup::loadPrevSave() {
00275     try{
00276         WarEngineMemento* saveFile = saveArchive->getLastSave();
00277         WarEngine::getInstance().loadSave(saveFile);
00278     }
00279     catch(const std::exception& error){
00280         std::cerr << error.what() << "\n";
00281     }
00282 }
00283 void EasySetup::loadSpecificSave(string name) {
00284     try{
00285         WarEngineMemento* saveFile = saveArchive->getSave(name);
00286         WarEngine::getInstance().loadSave(saveFile);
00287     }
00288     catch(const std::out_of_range& range_error){
00289         std::cerr << range_error.what() << "\n";
00290     }
00291 }
00292 }
00293
00294 #endif
00295
00296 #define EASYSETUP_H
00297 #include <iostream>

```

## 5.23 EasySetup.h

```

00001 #ifndef EASYSETUP_H
00002 #define EASYSETUP_H
00003 #include <iostream>

```

```

00004 #include <cctype>
00005 #include <string>
00006 #include <vector>
00007 #include "SaveArchive.h"
00008
00009 using namespace std;
00010
00011 class EasySetup
00012 {
00013     private:
00014         SaveArchive* saveArchive;
00015
00016     public:
00017         EasySetup();
00018         void setupSimulation();
00019         void runSimulation();
00020         void loadPrevSave();
00021         void loadSpecificSave(std::string name);
00022         void saveSimulationSetup();
00023 };
00024
00025 #endif

```

## 5.24 Entity.cpp

```

00001 #include "Entity.h"
00002 #include "Alliance.h"
00003
00004
00005 Entity::Entity() {
00006     health = 0;
00007     damage = 0;
00008     type = NULL;
00009 }
00010
00011 Entity::Entity(Type* type, int health, int damage) {
00012     this->health = health;
00013     this->damage = damage;
00014     this->type = type;
00015 }
00016
00017 Type* Entity::getType() {
00018     return this->type;
00019 }
00020
00021 void Entity::setType(Type* type) {
00022     this->type = type;
00023 }
00024
00025 Alliance* Entity::getAlliance() {
00026     return this->alliance;
00027 }
00028
00029 void Entity::setAlliance(Alliance* alliance) {
00030     this->alliance = alliance;
00031 }
00032
00033 int Entity::getHealth() {
00034     return this->health;
00035 }
00036
00037 void Entity::setHealth(int health) {
00038     this->health = health;
00039 }
00040
00041 int Entity::getDamage() {
00042     return this->damage;
00043 }
00044
00045 void Entity::setDamage(int damage) {
00046     this->damage = damage;
00047 }

```

## 5.25 Entity.h

```

00001 #ifndef ENTITY_H
00002 #define ENTITY_H
00003
00004 #include "Type.h"

```

```

00005
00006 class Alliance;
00007
00013 class Entity {
00014
00015 private:
00016     Type* type;
00017     Alliance* alliance;
00018     int health;
00019     int damage;
00020
00021 public:
00025     Entity();
00026
00032     Entity(Type* type, int health, int damage);
00033
00042     Type* getType();
00043
00056     void setType(Type* type);
00057
00066     Alliance* getAlliance();
00067
00080     void setAlliance(Alliance* alliance);
00081
00090     int getHealth();
00091
00104     void setHealth(int health);
00105
00114     int getDamage();
00115
00128     void setDamage(int damage);
00129
00130     virtual void takeDamage(int damage) = 0;
00131
00132     virtual void dealDamage(Entity* entity) = 0;
00133
00134     virtual Entity* clone() = 0;
00135 };
00136
00137 #endif

```

## 5.26 Factory.cpp

```

00001 #include "Factory.h"
00002
00003 Factory::Factory(Type* type, AddOn* addOn) {
00004     this->type = type;
00005     this->addOn = addOn;
00006 }
00007
00008 Factory::~Factory() {
00009     delete type;
00010     delete addOn;
00011 }
00012
00013 Type* Factory::getType() {
00014     return this->type;
00015 }
00016
00017 void Factory::setType(Type* type) {
00018     this->type = type;
00019 }
00020
00021 AddOn* Factory::getAddOn() {
00022     return this->addOn;
00023 }
00024
00025 void Factory::setAddOns(AddOn* addOn) {
00026     this->addOn = addOn;
00027 }

```

## 5.27 Factory.h

```

00001 #ifndef FACTORY_H
00002 #define FACTORY_H
00003
00004 #include "Type.h"
00005 #include "AddOn.h"
00006

```



```

00012 class Factory {
00013
00014 private:
00015     Type* type;
00016     AddOn* addOn;
00017
00018 public:
00025     Factory(Type* type, AddOn* addOn);
00026
00033     ~Factory();
00034
00035     virtual Entity* createEntity(Alliance* alliance) = 0;
00036
00045     Type* getType();
00046
00047
00060     void setType(Type* type);
00061
00062
00071     AddOn* getAddOn();
00072
00085     void setAddOns(AddOn* addOn);
00086
00087     virtual Factory* clone() = 0;
00088 };
00089
00090 #endif

```

## 5.28 General.cpp

```

00001 #include "General.h"
00002
00003 General::General(Alliance* alliance, Strategy* strategy) {
00004     this->alliance = alliance;
00005     this->strategy = strategy;
00006     numDeaths = 0;
00007 }
00008
00009 void General::initiateStrategy(KeyPoint* keyPoint) {
00010     numDeaths++;
00011     if (numDeaths >= 5) {
00012         strategy->performStrat(keyPoint, this->alliance);
00013         numDeaths = 0;
00014     }
00015 }
00016
00017 bool General::setStrategy(Strategy* strategy){
00018     this->strategy = strategy;
00019     return true;
00020 }
00021
00022 Alliance* General::getAlliance(){
00023     return this->alliance;
00024 }

```

## 5.29 General.h

```

00001 #ifndef GENERAL_H
00002 #define GENERAL_H
00003 #include "Alliance.h"
00004 #include "Strategy.h"
00005
00006 class KeyPoint;
00007
00008 class General {
00009
00010 private:
00011     Alliance* alliance;
00012     Strategy* strategy;
00013     int numDeaths;
00014
00015 public:
00022     General(Alliance* alliance, Strategy* strategy);
00023
00033     void initiateStrategy(KeyPoint* keyPoint);
00034
00049     bool setStrategy(Strategy* strategy);
00050
00059     Alliance* getAlliance();

```

```

00060 };
00061
00062 #endif

```

## 5.30 KeyPoint.cpp

```

00001 #include "KeyPoint.h"
00002 #include "Weather.h"
00003 #include "RoundStats.h"
00004 #include "Sunny.h"
00005 #include <time.h>
00006 #include <cstdlib>
00007 #include <iostream>
00008
00009 using namespace std;
00010
00011 KeyPoint::KeyPoint(string areaName): Area(areaName) {
00012     weather = new Sunny();
00013 }
00014
00015 KeyPoint::KeyPoint(KeyPoint& keyPoint): Area(keyPoint.getAreaName()) {
00016     for (int i = 0; i < keyPoint.entities.size(); i++)
00017         this->addEntity(keyPoint.entities[i]->clone());
00018     weather = keyPoint.weather->clone();
00019 }
00020
00021 KeyPoint::~KeyPoint() {
00022     for (int i = 0; i < entities.size(); i++)
00023         delete entities[i];
00024     for (int i = 0; i < generals.size(); i++)
00025         delete generals[i];
00026     delete weather;
00027 }
00028
00029 bool KeyPoint::isKeyPoint() {
00030     return true;
00031 }
00032
00033 void KeyPoint::simulateBattle(Alliance* alliance) {
00034     int numUnits = 0;
00035     for (int i = 0; i < entities.size(); i++) {
00036         if (entities[i]->getAlliance() == alliance) {
00037             numUnits++;
00038         }
00039     }
00040     if (numUnits != entities.size()) {
00041         for (int i = 0; i < entities.size(); i++) {
00042             if (entities[i]->getAlliance() == alliance) {
00043                 int random;
00044                 do {
00045                     random = rand() % entities.size();
00046                 } while (entities[random]->getAlliance() == alliance);
00047                 if (rand() % ((int)(weather->getMultiplier() * 100)) <= (int)(weather->getMultiplier() *
100))
00048                     entities[i]->dealDamage(entities[random]);
00049             }
00050         }
00051     }
00052     clearBattlefield(alliance);
00053 }
00054
00055 void KeyPoint::clearBattlefield(Alliance* alliance) {
00056     int destroyed = 0;
00057     double numUnits = 0;
00058     for (vector<Entity*>::iterator it = entities.begin(); it != entities.end(); ++it) {
00059         if ((*it)->getHealth() <= 0) {
00060             destroyed++;
00061             for (int i = 0; i < generals.size(); i++) {
00062                 if (generals[i]->getAlliance() == (*it)->getAlliance()) {
00063                     generals[i]->initiateStrategy(this);
00064                     delete *it;
00065                     entities.erase(it);
00066                 }
00067             }
00068         } else if ((*it)->getAlliance() == alliance) {
00069             numUnits++;
00070         }
00071     }
00072 }

```

```

00077     }
00078
00079     // saving stats
00080     string stats = getAreaName() + ":\n";
00081     stats += "Key Point Satus: ";
00082     if (numUnits / entities.size() >= 0.6) {
00083         stats += "Winning\n";
00084     } else if (numUnits / entities.size() >= 0.35) {
00085         stats += "Contested\n";
00086     } else {
00087         stats += "Losing\n";
00088     }
00089     stats += "Number of Entities Destroyed by Alliance: " + to_string(destroyed);
00090
00091     RoundStats::keyPointInformation.push_back(stats);
00092     RoundStats::numEntitiesDestroyed += destroyed;
00093 }
00094
00095 void KeyPoint::moveEntitiesInto(Alliance* alliance, int numTroops) {
00096     vector<Entity*> troops = alliance->getReserveEntities(numTroops);
00097     for (int i = 0; i < troops.size(); i++)
00098         entities.push_back(troops[i]);
00099 }
00100
00101 void KeyPoint::moveEntitiesOutOf(Alliance* alliance, int numTroops) {
00102     vector<Entity*>::iterator it = entities.begin();
00103     for (int i = 0; i < numTroops && it != entities.end(); i++) {
00104         for (; it != entities.end(); ++it) {
00105             if ((*it)->getAlliance() == alliance) {
00106                 alliance->addReserveEntity(*it);
00107                 entities.erase(it);
00108             }
00109         }
00110     }
00111 }
00112
00113 void KeyPoint::addEntity(Entity* entity) {
00114     entities.push_back(entity);
00115 }
00116
00117 void KeyPoint::addGeneral(General* general) {
00118     generals.push_back(general);
00119 }
00120
00121 void KeyPoint::removeGeneral(General* general) {
00122     for (vector<General*>::iterator it = generals.begin(); it != generals.end(); ++it) {
00123         if (*it == general) {
00124             delete *it;
00125             generals.erase(it);
00126             return;
00127         }
00128     }
00129 }
00130
00131 Area* KeyPoint::clone() {
00132     return new KeyPoint(*this);
00133 }
00134
00135 void KeyPoint::setWeather(Weather* weather) {
00136     delete this->weather;
00137     this->weather = weather;
00138 }
00139
00140 void KeyPoint::changeWeather() {
00141
00142     srand(time(0));
00143
00144     int randomNum = 1 + (rand() % 10);
00145     std::string currWeather = this->weather->getWeather();
00146
00147     if (currWeather == "Sunny" && randomNum > 6) // 60% chance of not changing weather from Sunny and
        staying
00148         this->weather->handleChange(this);
00149     else if (currWeather == "Cloudy" && randomNum > 3) // 30% chance of not changing weather from
        Cloudy and staying
00150         this->weather->handleChange(this);
00151     else if (currWeather == "Rainy" && randomNum > 1) // 10% chance of not changing weather from Rainy
        and staying
00152         this->weather->handleChange(this);
00153
00154
00155 }
00156
00157 std::string KeyPoint::getWeather() const {
00158     return this->weather->getWeather();
00159 }

```

## 5.31 KeyPoint.h

```

00001 #ifndef KEYPOINT_H
00002 #define KEYPOINT_H
00003
00004 #include "Alliance.h"
00005 #include "Area.h"
00006 #include "Entity.h"
00007 #include "General.h"
00008 #include <vector>
00009
00010 class Weather;
00011
00017 class KeyPoint : public Area {
00018
00019 private:
00020     vector<Entity*> entities;
00021     vector<General*> generals;
00022     Weather* weather;
00023     std::string AreaType;
00024
00025 public:
00031     KeyPoint(std::string areaName);
00032
00038     KeyPoint(KeyPoint& keyPoint);
00039
00040     ~KeyPoint();
00041
00050     bool isKeyPoint();
00051
00064     void simulateBattle(Alliance* alliance);
00065
00075     void clearBattlefield(Alliance* alliance);
00076
00091     void moveEntitiesInto(Alliance* alliance, int numTroops);
00092
00107     void moveEntitiesOutOf(Alliance* alliance, int numTroops);
00108
00121     void addEntity(Entity* entity);
00122
00123     void addGeneral(General* general);
00124
00125     void removeGeneral(General* general);
00126
00135     Area* clone();
00136
00141     void changeWeather();
00142
00155     void setWeather(Weather* weather);
00156
00162     std::string getWeather() const;
00163
00164 };
00165
00166 #endif

```

## 5.32 Negotiator.cpp

```

00001 #include "Negotiator.h"
00002 #include<bits/stdc++.h>
00003
00004 Negotiator::Negotiator() {}
00005
00006 Negotiator::~Negotiator() {
00007     alliances.clear();
00008 }
00009
00010 bool Negotiator::sendPeace(Alliance* offerAlliance) {
00011
00012     for (int yy = 0; yy < alliances.size(); yy++)
00013     {
00014         if (alliances[yy] != offerAlliance) {
00015             if (alliances[yy]->considerPeace() == false)
00016                 return false; // There is at least one enemy alliances that does not want the peace
00017         }
00018     }
00019
00020     return true; // All the alliances being fought against agreed to the peace deal
00021 }
00022
00023
00024 void Negotiator::removeAlliance(Alliance* oldAlliance) {

```

```

00025
00026     for (int xx = 0; xx < alliances.size(); xx++)
00027     {
00028         if (alliances[xx]->getID() == oldAlliance->getID())
00029             alliances.erase( alliances.begin() + xx ); // Removes the specific alliances from this
negotiator
00030     }
00031
00032 }
00033
00034 void Negotiator::addAlliance(Alliance* newAlliance) {
00035
00036     if (std::find(alliances.begin(), alliances.end(), newAlliance) != alliances.end())
00037         alliances.push_back(newAlliance);
00038
00039 }
00040
00041 int Negotiator::getNumAlliances() {
00042     return this->alliances.size();
00043 }

```

## 5.33 Negotiator.h

```

00001 #ifndef NEGOTIATOR_H
00002 #define NEGOTIATOR_H
00003 #include <vector>
00004 #include "Alliance.h"
00005
00006 class Negotiator {
00007 private:
00008     vector<Alliance*> alliances;
00009
00010 public:
00011     Negotiator();
00012     ~Negotiator();
00013
00014     bool sendPeace(Alliance* offerAlliance);
00015
00016     void removeAlliance(Alliance* oldAlliance);
00017
00018     void addAlliance(Alliance* newAlliance);
00019
00020     int getNumAlliances();
00021 };
00022
00023 #endif

```

## 5.34 NegotiatorTest.h

```

00001 #include <stdexcept>
00002 #include "Negotiator.h"
00003 #include "Alliance.h"
00004 #include "gtest/gtest.h"
00005
00006 namespace {
00007
00008     // Tests Negotiator Functionality
00009
00010     // Tests Negotiator offerPeace()
00011     // ===== Positive Testing =====
00012     // Test Preconditions Bounds
00013     TEST(NegotiatorOfferPeace, PositiveTesting) {
00014         Alliance* a = new Alliance();
00015         Alliance* b = new Alliance();
00016         Negotiator* n = new Negotiator();
00017         n->addAlliance(a);
00018         n->addAlliance(b);
00019         a->setNegotiator(n);
00020         b->setNegotiator(n);
00021
00022         if (a->offerPeace()) {
00023             EXPECT_EQ(3, a->getActive());
00024         } else {
00025             EXPECT_EQ(1, a->getActive());
00026         }
00027     }
00028 }

```

```

00029 // Tests Negotiator surrender()
00030 // ===== Positive Testing =====
00031 // Test Preconditions Bounds
00032 TEST(NegotiatorSurrender, PositiveTesting) {
00033     Alliance* a = new Alliance();
00034     Alliance* b = new Alliance();
00035     Alliance* c = new Alliance();
00036     Alliance* d = new Alliance();
00037     Alliance* e = new Alliance();
00038     Negotiator* n = new Negotiator();
00039     n->addAlliance(a);
00040     n->addAlliance(b);
00041     n->addAlliance(c);
00042     n->addAlliance(d);
00043     n->addAlliance(e);
00044     a->setNegotiator(n);
00045     b->setNegotiator(n);
00046     c->setNegotiator(n);
00047     d->setNegotiator(n);
00048     e->setNegotiator(n);
00049
00050     a->surrender();
00051     EXPECT_EQ(2, a->getActive());
00052
00053     b->surrender();
00054     EXPECT_EQ(2, a->getActive());
00055
00056     c->surrender();
00057     EXPECT_EQ(2, a->getActive());
00058
00059     d->surrender();
00060     EXPECT_EQ(2, a->getActive());
00061 }
00062
00063 }

```

## 5.35 Passive.cpp

```

00001 #include "Passive.h"
00002
00003 using namespace std;
00004
00005 Passive::Passive() {}
00006
00007 void Passive::performStrat(KeyPoint* keyPoint, Alliance* alliance) {
00008
00009     int randomNumber = (rand() % 10) + 5;
00010     keyPoint->moveEntitiesOutOf(alliance, randomNumber);
00011 }
00012
00013 Strategy* Passive::clone() {
00014     return new Passive();
00015 }

```

## 5.36 Passive.h

```

00001 #ifndef PASSIVE_H
00002 #define PASSIVE_H
00003 #include "Strategy.h"
00004 #include "KeyPoint.h"
00005
00006 class Passive : public Strategy {
00007
00008 public:
00009     Passive();
00010
00020     void performStrat(KeyPoint* keyPoint, Alliance* alliance);
00021
00027     Strategy* clone();
00028 };
00029
00030 #endif

```

## 5.37 Personnel.cpp

```

00001 #include "Personnel.h"

```

```

00002 #include "RoundStats.h"
00003 #include <iostream>
00004
00005 Personnel::Personnel(Type* type, int health, int damage): Entity(type, health, damage) {}
00006
00007 void Personnel::takeDamage(int damage) {
00008     setHealth(getHealth() - damage);
00009 }
00010
00011 void Personnel::dealDamage(Entity* entity) {
00012     RoundStats::damageDone += getDamage();
00013     entity->takeDamage(getDamage());
00014 }
00015
00016 Entity* Personnel::clone() {
00017     Personnel* p;
00018     if (this->getType() == NULL) {
00019         p = new Personnel(NULL, this->getHealth(), this->getDamage());
00020     } else {
00021         p = new Personnel(this->getType()->clone(), this->getHealth(), this->getDamage());
00022     }
00023
00024     p->setAlliance(this->getAlliance());
00025
00026     return p;
00027 }

```

## 5.38 Personnel.h

```

00001 #ifndef PERSONNEL_H
00002 #define PERSONNEL_H
00003
00004 #include "Entity.h"
00005
00011 class Personnel : public Entity {
00012
00013 public:
00021     Personnel(Type* type, int health = 100, int damage = 10);
00022
00035     void takeDamage(int damage);
00036
00049     void dealDamage(Entity* entity);
00050
00056     Entity* clone();
00057 };
00058
00059 #endif

```

## 5.39 PersonnelFactory.cpp

```

00001 #include "PersonnelFactory.h"
00002 #include "Personnel.h"
00003 #include <iostream>
00004
00005 PersonnelFactory::PersonnelFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00006
00007 Entity* PersonnelFactory::createEntity(Alliance* alliance) {
00008     Personnel* p = new Personnel(getType()->clone());
00009     p->setAlliance(alliance);
00010     if (getAddOn() != NULL) {
00011         AddOn* personnelAddOn = getAddOn()->clone();
00012         personnelAddOn->setEntity(p);
00013         return personnelAddOn;
00014     } else {
00015         return p;
00016     }
00017 }
00018
00019 Factory* PersonnelFactory::clone() {
00020     return new PersonnelFactory(getType()->clone(), getAddOn()->clone());
00021 }

```

## 5.40 PersonnelFactory.h

```

00001 #ifndef PERSONNELFACTORY_H

```

```

00002 #define PERSONNELFACTORY_H
00003
00004 #include "Factory.h"
00005
00011 class PersonnelFactory : public Factory {
00012
00013 public:
00020     PersonnelFactory(Type* type, AddOn* addOn);
00021
00034     Entity* createEntity(Alliance* alliance);
00035
00044     Factory* clone();
00045 };
00046
00047 #endif

```

## 5.41 Piercing.cpp

```

00001 #include "Piercing.h"
00002 #include "RoundStats.h"
00003
00004 Piercing::Piercing(int value) : AddOn(value) {}
00005
00006 void Piercing::takeDamage(int damage) {
00007     entity->takeDamage(damage);
00008 }
00009
00010 void Piercing::dealDamage(Entity* entity) {
00011     int sumValue = this->entity->getDamage() + value;
00012     entity->takeDamage(sumValue);
00013     RoundStats::damageDone += sumValue;
00014 }
00015
00016 AddOn* Piercing::clone() {
00017     Piercing* piercing = new Piercing(value);
00018     if (getEntity() != NULL)
00019         piercing->setEntity(entity->clone());
00020     return piercing;
00021 }

```

## 5.42 Piercing.h

```

00001 #ifndef PIERCING_H
00002 #define PIERCING_H
00003 #include "AddOn.h"
00004 #include "Entity.h"
00005
00011 class Piercing : public AddOn {
00012
00013 public:
00020     Piercing(int value);
00021
00034     void takeDamage(int damage);
00035
00048     void dealDamage(Entity* entity);
00049
00058     AddOn* clone();
00059 };
00060
00061 #endif

```

## 5.43 Rainy.cpp

```

00001 #include "Rainy.h"
00002 #include "Sunny.h"
00003
00004 Rainy::Rainy(): Weather() {
00005     this->multiplier = 0.5;
00006 }
00007
00008 std::string Rainy::getWeather() {
00009     return "Rainy";
00010 }
00011

```



```

00012 void Rainy::handleChange(KeyPoint* k) {
00013     Sunny* newWeather = new Sunny();
00014     k->setWeather(newWeather);
00015 }
00016
00017 Weather* Rainy::clone() {
00018     return new Rainy();
00019 }

```

## 5.44 Rainy.h

```

00001 #ifndef RAINY_H
00002 #define RAINY_H
00003 #include "Weather.h"
00004 #include "KeyPoint.h"
00005
00006 class Rainy : public Weather {
00007
00008
00009 public:
00013     Rainy();
00014
00023     std::string getWeather();
00024
00037     void handleChange(KeyPoint* k);
00038
00044     Weather* clone();
00045 };
00046
00047 #endif

```

## 5.45 RoundStats.cpp

```

00001 #include "RoundStats.h"
00002
00003 int RoundStats::numEntitiesCreated = 0;
00004 int RoundStats::numEntitiesDestroyed = 0;
00005 int RoundStats::damageDone = 0;
00006 vector<string> RoundStats::keyPointInformation;
00007 vector<string> RoundStats::entityMovementInformation;
00008
00009 void RoundStats::clearStats() {
00010     numEntitiesCreated = 0;
00011     numEntitiesDestroyed = 0;
00012     keyPointInformation.clear();
00013     entityMovementInformation.clear();
00014 }
00015
00016 string RoundStats::toString() {
00017     string out = "Number of Entities Created: " + to_string(numEntitiesCreated) + "\n";
00018     out += "Number of Entities Destroyed by Alliance: " + to_string(numEntitiesDestroyed) + "\n";
00019     out += "Damage Given by Alliance: " + to_string(damageDone) + "\n";
00020
00021     out += "\nKey Point Round Information:\n";
00022     for (int i = 0; i < keyPointInformation.size(); i++)
00023         out += keyPointInformation[i] + "\n";
00024
00025     out += "\nMovement Round Information:\n";
00026     for (int i = 0; i < entityMovementInformation.size(); i++)
00027         out += entityMovementInformation[i] + "\n";
00028
00029     return out;
00030 }

```

## 5.46 RoundStats.h

```

00001 #ifndef ROUNDSTATS_H
00002 #define ROUNDSTATS_H
00003
00004 #include <vector>
00005 #include <string>
00006
00007 using namespace std;
00008
00009 class RoundStats {

```

```

00010     public:
00011         static int numEntitiesCreated;
00012         static int numEntitiesDestroyed;
00013         static int damageDone;
00014         static vector<string> keyPointInformation;
00015         static vector<string> entityMovementInformation;
00016
00017         static void clearStats();
00018         static string toString();
00019 };
00020
00021 #endif

```

## 5.47 SaveArchive.cpp

```

00001 #include "SaveArchive.h"
00002
00003 SaveArchive::SaveArchive() {}
00004
00005 void SaveArchive::addNewSave(std::string newSaveName, WarEngineMemento* newSave) {
00006     saveList.insert({newSaveName, newSave});
00007 }
00008
00009 WarEngineMemento* SaveArchive::getLastSave() {
00010
00011     if(saveList.size() == 0){
00012         throw "Save archive is empty.";
00013     }
00014
00015     WarEngineMemento* lastSave = saveList.begin()->second;
00016
00017     saveList.erase( saveList.begin() );
00018
00019     return lastSave;
00020 }
00021
00022 WarEngineMemento* SaveArchive::getSave(std::string name) {
00023     if(saveList.size() == 0){
00024         std::__throw_out_of_range("Save archive is empty");
00025     }
00026
00027     auto iter = saveList.find(name);
00028
00029     if(iter == saveList.end())
00030         std::__throw_invalid_argument("No save with given name exists");
00031
00032     return iter->second;
00033 }
00034
00035 void SaveArchive::clearSaveList() {
00036     saveList.clear();
00037 }
00038
00039 void SaveArchive::deleteSave(std::string name) {
00040     if(saveList.size() == 0){
00041         std::__throw_out_of_range("Save archive is empty");
00042     }
00043
00044     auto iter = saveList.find(name) ;
00045
00046     if(iter == saveList.end())
00047         return;
00048
00049     saveList.erase( iter );
00050 }

```

## 5.48 SaveArchive.h

```

00001 #ifndef SAVEARCHIVE_H
00002 #define SAVEARCHIVE_H
00003 #include <unordered_map>
00004 #include <string>
00005 #include "WarEngineMemento.h"
00006
00011 class SaveArchive {
00012
00013 private:
00014     std::unordered_map<std::string, WarEngineMemento*> saveList;
00015

```

```

00016 public:
00020     SaveArchive();
00021
00035     void addNewSave(std::string newSaveName, WarEngineMemento* newSave);
00036
00047     WarEngineMemento* getLastSave();
00048
00063     WarEngineMemento* getSave(std::string name);
00064
00072     void clearSaveList();
00073
00087     void deleteSave(std::string name);
00088 };
00089
00090 #endif

```

## 5.49 Strategy.cpp

```

00001 #include "Strategy.h"
00002 #include "KeyPoint.h"
00003 #include "Alliance.h"
00004
00005 using namespace std;
00006
00007 Strategy::Strategy() {}
00008
00009 Strategy::~Strategy() {}
00010

```

## 5.50 Strategy.h

```

00001 #ifndef STRATEGY_H
00002 #define STRATEGY_H
00003 #include <string>
00004 #include <ctime>
00005 #include <cstdlib>
00006
00007 class KeyPoint;
00008 class Alliance;
00009
00010 class Strategy {
00011
00012 protected:
00013     std::string strategy;
00014
00015 public:
00020     Strategy();
00021
00026     ~Strategy();
00027
00028     virtual void performStrat(KeyPoint* keyPoint, Alliance* alliance) = 0;
00029
00030     virtual Strategy* clone() = 0;
00031 };
00032
00033 #endif

```

## 5.51 Sunny.cpp

```

00001 #include "Sunny.h"
00002 #include "Cloudy.h"
00003
00004 Sunny::Sunny() {
00005     this->multiplier = 1.0;
00006 }
00007
00008 std::string Sunny::getWeather() {
00009     return "Sunny";
00010 }
00011
00012 void Sunny::handleChange(KeyPoint* k) {
00013     Cloudy* newWeather = new Cloudy();
00014     k->setWeather(newWeather);
00015 }
00016
00017 Weather* Sunny::clone() {
00018     return new Sunny();
00019 }

```

## 5.52 Sunny.h

```

00001 #ifndef SUNNY_H
00002 #define SUNNY_H
00003 #include "Weather.h"
00004 #include "KeyPoint.h"
00005
00006 #include "Weather.h"
00007
00008 class Sunny : public Weather {
00009
00010 public:
00014     Sunny();
00015
00024     virtual std::string getWeather();
00025
00038     virtual void handleChange(KeyPoint* k);
00039
00045     Weather* clone();
00046 };
00047
00048 #endif

```

## 5.53 Support.cpp

```

00001 #include "Support.h"
00002 #include "RoundStats.h"
00003
00004 Support::Support(Type* type, int health, int damage): Entity(type, health, damage) {}
00005
00006 void Support::dealDamage(Entity* entity) {
00007     RoundStats::damageDone += getDamage();
00008     entity->takeDamage(getDamage());
00009 }
00010
00011 void Support::takeDamage(int damage) {
00012     this->setHealth(this->getHealth() - damage);
00013 }
00014
00015 Entity* Support::clone() {
00016     Support* s;
00017     if (this->getType() == NULL) {
00018         s = new Support(NULL, this->getHealth(), this->getDamage());
00019     } else {
00020         s = new Support(this->getType()->clone(), this->getHealth(), this->getDamage());
00021     }
00022
00023     s->setAlliance(this->getAlliance());
00024
00025     return s;
00026 }

```

## 5.54 Support.h

```

00001 #ifndef SUPPORT_H
00002 #define SUPPORT_H
00003
00004 #include "Entity.h"
00005
00011 class Support : public Entity {
00012
00013 public:
00021     Support(Type* type, int health = 1000, int damage = 30);
00022
00035     void takeDamage(int damage);
00036
00049     void dealDamage(Entity* entity);
00050
00056     Entity* clone();
00057 };
00058
00059 #endif

```

## 5.55 SupportFactory.cpp

```

00001 #include "SupportFactory.h"

```

```

00002 #include "Support.h"
00003
00004 SupportFactory::SupportFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00005
00006 Entity* SupportFactory::createEntity(Alliance* alliance) {
00007     Support* s = new Support(getType()->clone());
00008     s->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(s);
00012         return personnelAddOn;
00013     } else {
00014         return s;
00015     }
00016 }
00017
00018 Factory* SupportFactory::clone() {
00019     return new SupportFactory(getType()->clone(), getAddOn()->clone());
00020 }

```

## 5.56 SupportFactory.h

```

00001 #ifndef SUPPORTFACTORY_H
00002 #define SUPPORTFACTORY_H
00003
00004 #include "Factory.h"
00005
00011 class SupportFactory : public Factory {
00012
00013 public:
00020     SupportFactory(Type* type, AddOn* addOn);
00021
00034     Entity* createEntity(Alliance* alliance);
00035
00044     Factory* clone();
00045 };
00046
00047 #endif

```

## 5.57 TerrainType.cpp

```

00001 #include "TerrainType.h"
00002
00003 TerrainType::TerrainType() {}
00004
00005 string TerrainType::getTypeDesc() {
00006     return "Terrain";
00007 }
00008
00009 Type* TerrainType::clone() {
00010     return new TerrainType();
00011 }

```

## 5.58 TerrainType.h

```

00001 #ifndef TERRAINTYPE_H
00002 #define TERRAINTYPE_H
00003
00004 #include "Type.h"
00005
00011 class TerrainType : public Type {
00012
00013 public:
00017     TerrainType();
00018
00027     string getTypeDesc();
00028
00029     Type* clone();
00030 };
00031
00032 #endif

```

## 5.59 testmain.cpp

```

00001 #include "NegotiatorTest.h"
00002 #include "AddOnTest.h"
00003 #include <gtest/gtest.h>
00004 #include "EasySetup.h"
00005 #include "WarEngine.h"
00006 #include "KeyPoint.h"
00007
00008 void setupWarEngine() {
00009     Alliance* a1 = new Alliance();
00010     a1->addCountry(new Country("Germany"));
00011
00012     Alliance* a2 = new Alliance();
00013     a2->addCountry(new Country("Finland"));
00014
00015     WarEngine::getInstance().addAlliance(a1);
00016     WarEngine::getInstance().addAlliance(a2);
00017
00018     KeyPoint* k1 = new KeyPoint("West");
00019     KeyPoint* k2 = new KeyPoint("North");
00020     KeyPoint* k3 = new KeyPoint("East");
00021
00022     Personnel* p1 = new Personnel(NULL);
00023     p1->setAlliance(a1);
00024     k1->addEntity(p1->clone());
00025     k2->addEntity(p1->clone());
00026     k3->addEntity(p1->clone());
00027
00028     Personnel* p2 = new Personnel(NULL);
00029     p2->setAlliance(a2);
00030     k1->addEntity(p2->clone());
00031     k2->addEntity(p2->clone());
00032
00033     WarTheatre* w = new WarTheatre("Europe");
00034     w->addArea(k1);
00035     w->addArea(k2);
00036     w->addArea(k3);
00037
00038     WarEngine::getInstance().setWarTheatre(w);
00039 }
00040
00041 int main(int argc, char **argv) {
00042     // EasySetup* easySetup = new EasySetup();
00043     // easySetup->setupSimulation();
00044
00045     setupWarEngine();
00046     WarEngine::getInstance().simulate();
00047
00048     // testing::InitGoogleTest(&argc, argv);
00049     // return RUN_ALL_TESTS();
00050 }

```

## 5.60 Type.cpp

```

00001 #include "Type.h"
00002
00003 Type::Type() {}

```

## 5.61 Type.h

```

00001 #ifndef TYPE_H
00002 #define TYPE_H
00003
00004 #include <string>
00005
00006 using namespace std;
00007
00013 class Type {
00014
00015 public:
00019     Type();
00020
00021     virtual string getTypeDesc() = 0;
00022
00023     virtual Type* clone() = 0;
00024
00025 };
00026
00027 #endif

```

## 5.62 Vehicle.cpp

```

00001 #include "Vehicle.h"
00002 #include "RoundStats.h"
00003
00004 Vehicle::Vehicle(Type* type, int health, int damage): Entity(type, health, damage) {}
00005
00006 void Vehicle::takeDamage(int damage) {
00007     setHealth(getHealth() - damage);
00008 }
00009
00010 void Vehicle::dealDamage(Entity* entity) {
00011     RoundStats::damageDone += getDamage();
00012     entity->takeDamage(getDamage());
00013 }
00014
00015 Entity* Vehicle::clone() {
00016     Vehicle* v;
00017     if (this->getType() == NULL) {
00018         v = new Vehicle(NULL, this->getHealth(), this->getDamage());
00019     } else {
00020         v = new Vehicle(this->getType()->clone(), this->getHealth(), this->getDamage());
00021     }
00022     v->setAlliance(this->getAlliance());
00023     return v;
00024 }
00025
00026 }
```

## 5.63 Vehicle.h

```

00001 #ifndef VEHICLE_H
00002 #define VEHICLE_H
00003
00004 #include "Entity.h"
00005
00011 class Vehicle : public Entity {
00012 public:
00021     Vehicle(Type* type, int health = 500, int damage = 10);
00022
00035     void takeDamage(int damage);
00036
00049     void dealDamage(Entity* entity);
00050
00056     Entity* clone();
00057 };
00058
00059 #endif
```

## 5.64 VehicleFactory.cpp

```

00001 #include "VehicleFactory.h"
00002 #include "Vehicle.h"
00003
00004 VehicleFactory::VehicleFactory(Type* type, AddOn* addOn): Factory(type, addOn) {}
00005
00006 Entity* VehicleFactory::createEntity(Alliance* alliance) {
00007     Vehicle* v = new Vehicle(getType()->clone());
00008     v->setAlliance(alliance);
00009     if (getAddOn() != NULL) {
00010         AddOn* personnelAddOn = getAddOn()->clone();
00011         personnelAddOn->setEntity(v);
00012         return personnelAddOn;
00013     } else {
00014         return v;
00015     }
00016 }
00017
00018 Factory* VehicleFactory::clone() {
00019     return new VehicleFactory(getType()->clone(), getAddOn()->clone());
00020 }
```

## 5.65 VehicleFactory.h

```

00001 #ifndef VEHICLEFACTORY_H
```

```

00002 #define VEHICLEFACTORY_H
00003 #include "Factory.h"
00004
00010 class VehicleFactory : public Factory {
00011
00012 public:
00019     VehicleFactory(Type* type, AddOn* addOn);
00020
00033     Entity* createEntity(Alliance* alliance);
00034
00043     Factory* clone();
00044 };
00045
00046 #endif

```

## 5.66 WarEngine.cpp

```

00001 #include "WarEngine.h"
00002 #include "RoundStats.h"
00003 #include <iostream>
00004
00005 WarEngine::WarEngine() {
00006     this->state = new WarEngineState();
00007 }
00008
00009 WarEngineMemento* WarEngine::saveState() {
00010     return new WarEngineMemento(state->clone());
00011 }
00012
00013 void WarEngine::loadSave(WarEngineMemento* save) {
00014     delete this->state;
00015     this->state = save->getState();
00016 }
00017
00018 WarEngine& WarEngine::getInstance() {
00019     static WarEngine uniqueInstance_;
00020     return uniqueInstance_;
00021 }
00022
00023 WarEngine::~WarEngine() {
00024     delete this->state;
00025 }
00026
00027
00028 void WarEngine::simulate() {
00029     vector<Alliance*> alliances = this->state->getAlliances();
00030
00031     for(int i = 0; i < alliances.size(); i++) {
00032         if (alliances[i]->getActive() == 1) {
00033             RoundStats::clearStats();
00034             state->getArea()->simulateBattle(alliances[i]);
00035             cout << "===== " << endl;
00036             cout << "Alliance " << alliances[i]->getID() << ":" << endl;
00037
00038             if (alliances[i]->getID() == 2) {
00039                 cout << "Status: Surrendered" << endl;
00040             } else if (alliances[i]->getID() == 3) {
00041                 cout << "Status: Found Peace" << endl;
00042             } else {
00043                 cout << "Status: Active" << endl;
00044             }
00045
00046             cout << RoundStats::toString() << endl;
00047             cout << "===== " << endl;
00048         }
00049     }
00050
00051 }
00052
00053 void WarEngine::setWarTheatre(WarTheatre* battleGround) {
00054     state->setArea(battleGround);
00055 }
00056
00057 void WarEngine::addAlliance(Alliance* alliance) {
00058     state->alliances.push_back(alliance);
00059 }

```

## 5.67 WarEngine.h

```

00001 #ifndef WARENGINE_H

```



```

00002 #define WAREENGINE_H
00003
00004 #include "WarEngineState.h"
00005 #include "WarEngineMemento.h"
00006 #include "WarTheatre.h"
00007
00012 class WarEngine {
00013
00014 private:
00015     WarEngineState* state;
00016     bool gameOver;
00017
00018 protected:
00023     WarEngine();
00024
00033     WarEngine(const WarEngine&){};
00034
00038     WarEngine& operator=(const WarEngine&){ return *this; };
00039
00044     ~WarEngine();
00045
00046 public:
00051     WarEngineMemento* saveState();
00052
00065     void loadSave(WarEngineMemento* save);
00066
00072     static WarEngine& getInstance();
00073
00087     void setWarTheatre(WarTheatre* battleGround);
00088
00089     void addAlliance(Alliance* alliance);
00090
00099     void simulate();
00100
00101 };
00102
00103 #endif

```

## 5.68 WarEngineMemento.cpp

```

00001 #include "WarEngineMemento.h"
00002
00003 WarEngineMemento::WarEngineMemento(WarEngineState * state){
00004     this->state = state;
00005 }
00006
00007 void WarEngineMemento::setState(WarEngineState* state){
00008     this->state = state;
00009 }
00010
00011 WarEngineState* WarEngineMemento::getState(){
00012     return state;
00013 }

```

## 5.69 WarEngineMemento.h

```

00001 #ifndef WAREENGINEMEMENTO_H
00002 #define WAREENGINEMEMENTO_H
00003
00004 #include "WarEngineState.h"
00005 #include <string>
00006 #include <vector>
00007
00008 class WarEngine;
00009
00015 class WarEngineMemento {
00016
00017 friend class WarEngine;
00018
00019 private:
00020     WarEngineState* state;
00021
00028     WarEngineMemento(WarEngineState* state);
00029
00039     void setState(WarEngineState* state);
00040
00048     WarEngineState* getState();
00049
00050 };
00051
00052 #endif

```

## 5.70 WarEngineState.cpp

```

00001 #include "WarEngineState.h"
00002 #include "Negotiator.h"
00003 #include "Defensive.h"
00004
00005 WarEngineState::WarEngineState() {
00006     area = nullptr;
00007 }
00008
00009 void WarEngineState::setArea(Area* area) {
00010     this->area = area;
00011 }
00012
00013 Area* WarEngineState::getArea() {
00014
00015     if(area == nullptr)
00016         throw "No Areas Stored.";
00017
00018     return this->area;
00019 }
00020
00021 void WarEngineState::setAlliances(vector<Alliance*> alliances) {
00022     this->alliances = alliances;
00023 }
00024
00025 vector<Alliance*> WarEngineState::getAlliances() {
00026
00027     if(alliances.size() == 0)
00028         std::__throw_out_of_range("No Alliances stored.");
00029
00030     return alliances;
00031 }
00032
00033 WarEngineState* WarEngineState::clone() {
00034
00035     WarEngineState* clonedState = new WarEngineState();
00036     Area* copiedArea = this->area->clone();
00037
00038     Negotiator* negotiator = new Negotiator();
00039     for(Alliance* alliance : this->alliances){
00040
00041         Alliance* clonedAlliance = alliance->clone();
00042
00043         clonedState->alliances.push_back(clonedAlliance);
00044         clonedAlliance->setNegotiator(negotiator);
00045         negotiator->addAlliance(clonedAlliance);
00046         copiedArea->addGeneral(new General(clonedAlliance, new Defensive()));
00047     }
00048
00049     clonedState->setArea(copiedArea);
00050
00051     return clonedState;
00052 }
00053
00054 WarEngineState::~WarEngineState(){
00055
00056     for(Alliance* alliance : this->alliances){
00057         delete alliance;
00058     }
00059
00060     //delete this->area;
00061
00062 }

```

## 5.71 WarEngineState.h

```

00001 #ifndef WARENGINESTATE_H
00002 #define WARENGINESTATE_H
00003 #include "Alliance.h"
00004 #include "Area.h"
00005 #include <vector>
00006
00007 class WarEngine;
00008
00009 using namespace std;
00010
00011 class WarEngineState {
00012
00013     friend class WarEngine;
00014
00015 private:
00016     Area* area;

```

```

00023     vector<Alliance*> alliances;
00024
00029     WarEngineState();
00030
00043     void setArea(Area* area);
00044
00053     Area* getArea();
00054
00067     void setAlliances(vector<Alliance*> alliances);
00068
00077     vector<Alliance*> getAlliances();
00078
00084     WarEngineState* clone();
00085
00089     ~WarEngineState();
00090 };
00091
00092 #endif

```

## 5.72 WarTheatre.cpp

```

00001 #include "WarTheatre.h"
00002
00003 using namespace std;
00004
00005 WarTheatre::WarTheatre(string areaName): Area(areaName) {}
00006
00007 WarTheatre::~WarTheatre() {
00008     for (int i = 0; i < areas.size(); i++)
00009         delete areas[i];
00010 }
00011
00012 bool WarTheatre::isKeyPoint() {
00013     return false;
00014 }
00015
00016 void WarTheatre::simulateBattle(Alliance* alliance) {
00017     for (int i = 0; i < areas.size(); i++)
00018         areas[i]->simulateBattle(alliance);
00019 }
00020
00021 void WarTheatre::addArea(Area* area) {
00022     areas.push_back(area);
00023 }
00024
00025 WarTheatre* WarTheatre::clone() {
00026     WarTheatre* w = new WarTheatre(getAreaName());
00027
00028     for (int i = 0; i < areas.size(); i++)
00029         w->addArea(areas[i]->clone());
00030
00031     return w;
00032 }
00033
00034 void WarTheatre::addGeneral(General* general) {
00035     for (int i = 0; i < areas.size(); i++)
00036         areas[i]->addGeneral(general);
00037 }

```

## 5.73 WarTheatre.h

```

00001 #ifndef WARTHEATRE_H
00002 #define WARTHEATRE_H
00003
00004 #include "Area.h"
00005 #include "Alliance.h"
00006 #include <vector>
00007
00008 using namespace std;
00009
00010 class WarTheatre : public Area {
00011 private:
00012     vector<Area*> areas;
00013 public:
00014     WarTheatre(std::string areaName);
00015     ~WarTheatre();

```

```
00028
00037     bool isKeyPoint();
00038
00051     void simulateBattle(Alliance* alliance);
00052
00065     void addArea(Area* area);
00066
00078     void addGeneral(General* general);
00079
00088     WarTheatre* clone();
00089 };
00090
00091 #endif
```

## 5.74 Weather.cpp

```
00001 #include "Weather.h"
00002
00003 Weather::Weather() {}
00004
00005 Weather::~Weather() {}
00006
00007 double Weather::getMultiplier() {
00008     return this->multiplier;
00009 }
```

## 5.75 Weather.h

```
00001 #ifndef WEATHER_H
00002 #define WEATHER_H
00003 #include <string>
00004 #include "Weather.h"
00005 #include "KeyPoint.h"
00006
00007 class KeyPoint;
00008
00009 class Weather {
00010
00011 protected:
00012     double multiplier;
00013
00014 public:
00018     Weather();
00019
00023     ~Weather();
00024
00033     double getMultiplier();
00034
00035     virtual void handleChange(KeyPoint* k) = 0;
00036
00037     virtual std::string getWeather() = 0;
00038
00039     virtual Weather* clone() = 0;
00040 };
00041
00042 #endif
```

# Index

- ~Alliance
  - Alliance, [16](#)
- ~Area
  - Area, [25](#)
- ~Factory
  - Factory, [48](#)
- ~KeyPoint
  - KeyPoint, [55](#)
- ~Negotiator
  - Negotiator, [63](#)
- ~Strategy
  - Strategy, [85](#)
- ~WarEngine
  - WarEngine, [103](#)
- ~WarTheatre
  - WarTheatre, [109](#)
- ~Weather
  - Weather, [113](#)
- addAlliance
  - Negotiator, [63](#)
  - WarEngine, [104](#)
- addArea
  - WarTheatre, [109](#)
- addCountry
  - Alliance, [17](#)
- addEntity
  - KeyPoint, [56](#)
- addFactory
  - Alliance, [17](#)
- addGeneral
  - Area, [26](#)
  - KeyPoint, [56](#)
  - WarTheatre, [110](#)
- addNewSave
  - SaveArchive, [81](#)
- AddOn, [7](#)
  - AddOn, [8](#)
  - clone, [8](#)
  - dealDamage, [8](#)
  - entity, [11](#)
  - getEntity, [8](#)
  - getValue, [9](#)
  - setEntity, [9](#)
  - setValue, [10](#)
  - takeDamage, [10](#)
  - value, [11](#)
- addReserveEntity
  - Alliance, [18](#)
- AerialType, [11](#)
  - AerialType, [12](#)
  - clone, [12](#)
  - getTypeDesc, [12](#)
- Aggressive, [13](#)
  - Aggressive, [13](#)
  - clone, [13](#)
  - performStrat, [14](#)
- Alliance, [15](#)
  - ~Alliance, [16](#)
  - addCountry, [17](#)
  - addFactory, [17](#)
  - addReserveEntity, [18](#)
  - Alliance, [15](#), [16](#)
  - clone, [18](#)
  - considerPeace, [19](#)
  - getActive, [19](#)
  - getID, [19](#)
  - getReserveEntities, [20](#)
  - numRemainingEntities, [21](#)
  - offerPeace, [21](#)
  - runFactories, [21](#)
  - setActiveStatus, [21](#)
  - setNegotiator, [22](#)
  - surrender, [22](#)
- AquaticType, [23](#)
  - AquaticType, [24](#)
  - clone, [24](#)
  - getTypeDesc, [24](#)
- Area, [25](#)
  - ~Area, [25](#)
  - addGeneral, [26](#)
  - Area, [25](#)
  - clone, [26](#)
  - getAreaName, [26](#)
  - isKeyPoint, [26](#)
  - simulateBattle, [27](#)
- Armour, [27](#)
  - Armour, [28](#)
  - clone, [28](#)
  - dealDamage, [28](#)
  - takeDamage, [29](#)
- changeWeather
  - KeyPoint, [56](#)
- clearBattlefield
  - KeyPoint, [57](#)
- clearSaveList
  - SaveArchive, [81](#)
- clearStats
  - RoundStats, [78](#)

- clone
  - AddOn, 8
  - AerialType, 12
  - Aggressive, 13
  - Alliance, 18
  - AquaticType, 24
  - Area, 26
  - Armour, 28
  - Cloudy, 31
  - Country, 33
  - Defensive, 35
  - Entity, 43
  - Factory, 48
  - KeyPoint, 58
  - Passive, 66
  - Personnel, 68
  - PersonnelFactory, 71
  - Piercing, 73
  - Rainy, 76
  - Strategy, 85
  - Sunny, 87
  - Support, 89
  - SupportFactory, 92
  - TerrainType, 94
  - Vehicle, 98
  - VehicleFactory, 101
  - WarTheatre, 110
  - Weather, 113
- Cloudy, 30
  - clone, 31
  - Cloudy, 31
  - getWeather, 31
  - handleChange, 31
- considerPeace
  - Alliance, 19
- Country, 32
  - clone, 33
  - Country, 33
  - getID, 33
  - getName, 34
- createEntity
  - Factory, 49
  - PersonnelFactory, 71
  - SupportFactory, 93
  - VehicleFactory, 101
- damageDone
  - RoundStats, 79
- dealDamage
  - AddOn, 8
  - Armour, 28
  - Entity, 43
  - Personnel, 68
  - Piercing, 74
  - Support, 90
  - Vehicle, 98
- Defensive, 34
  - clone, 35
  - Defensive, 35
  - performStrat, 35
- deleteSave
  - SaveArchive, 82
- EasySetup, 36
  - EasySetup, 36
  - loadPrevSave, 37
  - loadSpecificSave, 37
  - runSimulation, 37
  - saveSimulationSetup, 37
  - setupSimulation, 38
- Entity, 41
  - clone, 43
  - dealDamage, 43
  - Entity, 42
  - getAlliance, 43
  - getDamage, 43
  - getHealth, 44
  - getType, 44
  - setAlliance, 44
  - setDamage, 45
  - setHealth, 46
  - setType, 46
  - takeDamage, 47
- entity
  - AddOn, 11
- entityMovementInformation
  - RoundStats, 79
- Factory, 47
  - ~Factory, 48
  - clone, 48
  - createEntity, 49
  - Factory, 48
  - getAddOn, 49
  - getType, 49
  - setAddOns, 49
  - setType, 50
- General, 51
  - General, 51
  - getAlliance, 52
  - initiateStrategy, 52
  - setStrategy, 53
- getActive
  - Alliance, 19
- getAddOn
  - Factory, 49
- getAlliance
  - Entity, 43
  - General, 52
- getAreaName
  - Area, 26
- getDamage
  - Entity, 43
- getEntity
  - AddOn, 8
- getHealth
  - Entity, 44

- getID
  - Alliance, 19
  - Country, 33
- getInstance
  - WarEngine, 104
- getLastSave
  - SaveArchive, 82
- getMultiplier
  - Weather, 113
- getName
  - Country, 34
- getNumAlliances
  - Negotiator, 63
- getReserveEntities
  - Alliance, 20
- getSave
  - SaveArchive, 83
- getType
  - Entity, 44
  - Factory, 49
- getTypeDesc
  - AerialType, 12
  - AquaticType, 24
  - TerrainType, 95
  - Type, 96
- getValue
  - AddOn, 9
- getWeather
  - Cloudy, 31
  - KeyPoint, 58
  - Rainy, 76
  - Sunny, 87
  - Weather, 114
- handleChange
  - Cloudy, 31
  - Rainy, 77
  - Sunny, 87
  - Weather, 114
- initiateStrategy
  - General, 52
- isKeyPoint
  - Area, 26
  - KeyPoint, 58
  - WarTheatre, 111
- KeyPoint, 54
  - ~KeyPoint, 55
  - addEntity, 56
  - addGeneral, 56
  - changeWeather, 56
  - clearBattlefield, 57
  - clone, 58
  - getWeather, 58
  - isKeyPoint, 58
  - KeyPoint, 55
  - moveEntitiesInto, 59
  - moveEntitiesOutOf, 60
  - removeGeneral, 60
  - setWeather, 61
  - simulateBattle, 61
- keyPointInformation
  - RoundStats, 79
- loadPrevSave
  - EasySetup, 37
- loadSave
  - WarEngine, 104
- loadSpecificSave
  - EasySetup, 37
- moveEntitiesInto
  - KeyPoint, 59
- moveEntitiesOutOf
  - KeyPoint, 60
- multiplier
  - Weather, 114
- Negotiator, 62
  - ~Negotiator, 63
  - addAlliance, 63
  - getNumAlliances, 63
  - Negotiator, 63
  - removeAlliance, 64
  - sendPeace, 64
- numEntitiesCreated
  - RoundStats, 79
- numEntitiesDestroyed
  - RoundStats, 79
- numRemainingEntities
  - Alliance, 21
- offerPeace
  - Alliance, 21
- operator=
  - WarEngine, 105
- Passive, 65
  - clone, 66
  - Passive, 66
  - performStrat, 66
- performStrat
  - Aggressive, 14
  - Defensive, 35
  - Passive, 66
  - Strategy, 85
- Personnel, 67
  - clone, 68
  - dealDamage, 68
  - Personnel, 68
  - takeDamage, 69
- PersonnelFactory, 70
  - clone, 71
  - createEntity, 71
  - PersonnelFactory, 71
- Piercing, 72
  - clone, 73

- dealDamage, 74
- Piercing, 73
- takeDamage, 74
- Rainy, 75
  - clone, 76
  - getWeather, 76
  - handleChange, 77
  - Rainy, 76
- removeAlliance
  - Negotiator, 64
- removeGeneral
  - KeyPoint, 60
- RoundStats, 78
  - clearStats, 78
  - damageDone, 79
  - entityMovementInformation, 79
  - keyPointInformation, 79
  - numEntitiesCreated, 79
  - numEntitiesDestroyed, 79
  - toString, 78
- runFactories
  - Alliance, 21
- runSimulation
  - EasySetup, 37
- SaveArchive, 80
  - addNewSave, 81
  - clearSaveList, 81
  - deleteSave, 82
  - getLastSave, 82
  - getSave, 83
  - SaveArchive, 80
- saveSimulationSetup
  - EasySetup, 37
- saveState
  - WarEngine, 105
- sendPeace
  - Negotiator, 64
- setActiveStatus
  - Alliance, 21
- setAddOns
  - Factory, 49
- setAlliance
  - Entity, 44
- setDamage
  - Entity, 45
- setEntity
  - AddOn, 9
- setHealth
  - Entity, 46
- setNegotiator
  - Alliance, 22
- setStrategy
  - General, 53
- setType
  - Entity, 46
  - Factory, 50
- setupSimulation
  - EasySetup, 38
- setValue
  - AddOn, 10
- setWarTheatre
  - WarEngine, 105
- setWeather
  - KeyPoint, 61
- simulate
  - WarEngine, 106
- simulateBattle
  - Area, 27
  - KeyPoint, 61
  - WarTheatre, 111
- Strategy, 84
  - ~Strategy, 85
  - clone, 85
  - performStrat, 85
  - Strategy, 85
  - strategy, 85
- strategy
  - Strategy, 85
- Sunny, 86
  - clone, 87
  - getWeather, 87
  - handleChange, 87
  - Sunny, 86
- Support, 88
  - clone, 89
  - dealDamage, 90
  - Support, 89
  - takeDamage, 91
- SupportFactory, 91
  - clone, 92
  - createEntity, 93
  - SupportFactory, 92
- surrender
  - Alliance, 22
- takeDamage
  - AddOn, 10
  - Armour, 29
  - Entity, 47
  - Personnel, 69
  - Piercing, 74
  - Support, 91
  - Vehicle, 99
- TerrainType, 94
  - clone, 94
  - getTypeDesc, 95
  - TerrainType, 94
- toString
  - RoundStats, 78
- Type, 95
  - getTypeDesc, 96
  - Type, 96
- value
  - AddOn, 11
- Vehicle, 97



- clone, [98](#)
- dealDamage, [98](#)
- takeDamage, [99](#)
- Vehicle, [97](#)
- VehicleFactory, [100](#)
  - clone, [101](#)
  - createEntity, [101](#)
  - VehicleFactory, [100](#)
- WarEngine, [102](#)
  - ~WarEngine, [103](#)
  - addAlliance, [104](#)
  - getInstance, [104](#)
  - loadSave, [104](#)
  - operator=, [105](#)
  - saveState, [105](#)
  - setWarTheatre, [105](#)
  - simulate, [106](#)
  - WarEngine, [103](#)
  - WarEngineMemento, [107](#)
  - WarEngineState, [108](#)
- WarEngineMemento, [107](#)
  - WarEngine, [107](#)
- WarEngineState, [107](#)
  - WarEngine, [108](#)
- WarTheatre, [108](#)
  - ~WarTheatre, [109](#)
  - addArea, [109](#)
  - addGeneral, [110](#)
  - clone, [110](#)
  - isKeyPoint, [111](#)
  - simulateBattle, [111](#)
  - WarTheatre, [109](#)
- Weather, [112](#)
  - ~Weather, [113](#)
  - clone, [113](#)
  - getMultiplier, [113](#)
  - getWeather, [114](#)
  - handleChange, [114](#)
  - multiplier, [114](#)
  - Weather, [113](#)