**Modern Kriegsspiel Rules: Fourth Edition (2021)**

Kriegsspiel is a tabletop wargame originating in 19th-century Prussia as a tool to teach officer cadets tactics and strategy before leading troops into battle. Its rules, originally written by George von Reisswitz and later modified by Erich von Tchischwitz, are incredibly complex, with the purpose of recreating as realistically as possible the battlefields of the time. Kriegsspiel is the ancestor of all modern tabletop wargames, including the ever-popular *Warhammer 40,000*.

Upon rediscovering the old Kriegsspiel manuals in 2018, I wrote more modern rules aiming to recreate the spirit of the original while making it easier to play. It retains its original usefulness. The First Edition (2018), only worked for land battles, but could be adapted to any time and place. By the Second Edition, naval battles had been added, and the 2020 Third Edition included land, sea, and air, effectively uniting all three theaters of war.

This Fourth Edition is designed to be used with the Kriegsspiel program I have designed called Umpire, which is written in Python for the use of the umpire. The Fourth Edition achieves the original ultimate goal of making Kriegsspiel time-independent, place-independent, player number-independent, relatively easy to play, and fully immersive.

# Basic Mechanics and Principles

## Players and the Umpire

A Kriegsspiel campaign can be played with at least three players. At minimum, there must be two teams, or militaries, represented by at least one player each, plus an umpire to oversee the game. Militaries may be unified, with one or more players working together and with perfect information, or may be divided, where each player has a subcommand, and even players on a single team do not work together. Unified campaigns are ideal for new players.

The umpire is a universal; every campaign will have an umpire. The umpire’s role is to manage the gamespace and track the attributes of each unit. The Kriegsspiel program is designed to eliminate the use of paper-and-pencil by the umpire, and to prevent preferential treatment or cheating on his part.

In contrast to many other tabletop wargames, the players themselves do not touch or alter the gamespace itself. Only the umpire may alter the gamespace, including the battlefield, units, and any other objects. In any dispute over location or position, the umpire’s word is final.

## Gameplay

Gameplay of Fourth Edition Kriegsspiel is open-ended. Campaigns are prepared in advance, either based on real or imaginary scenarios. Campaigns will include the units each military will have, the structure of the battlefield, and the scenarios in which each military will win or lose. See the section “Creating a Gamefile” for more details on what goes into a particular campaign.

Gameplay is turn- and command-based. At the beginning of the campaign, a particular military will go first, either chosen at random by the umpire or based on the campaign scenario itself. During a player’s turn, they will issue a series of commands to their units. This is done by passing the commands to the umpire silently, either on paper or over a messaging system (the particular manner of delivering commands is irrelevant, provided that the opposing military cannot become aware of them). Commands submitted are final, and the umpire will interpret them as he believes it would realistically be interpreted on a real battlefield. Some units may receive more than one command, but only in a certain order, so it is imperative that the player issue them in the correct sequence. There is no limit to the number of commands that can be issued in a given turn, except as provided for by the attributes of the units and gamespace.

In campaigns involving air warfare, a separate air warfare turn is taken before other commands. In this turn, only air commands can be passed.

## Gamespace vs. Units

The gamespace itself consists of the *battlefield*, and the *structures.* The battlefield is composed of the actual natural terrain, whether it be land terrain or the depth and nature of the water, in the case of naval battles. The structures are any manmade thing on the gamespace, including buildings, fortifications, minefields, roads, bridges, and so on. Elements of the gamespace can be either *extrinsic*, that is, built by the players in the course of the game, or *intrinsic*, pre-existing the campaign.

|  |  |  |
| --- | --- | --- |
|  | **Intrinsic** | **Extrinsic** |
| **Battlefield** | Terrain | Terrain changed by the players (damming a river, clearing a forest, etc.) |
| **Structures** | Pre-existing towns, roads, etc. | Fortifications, rebuilt structures. |

The elements of the gamespace can be controlled by the players, but only through commands issued to *units.* Units are groups of soldiers, sailors, marines, or airmen, along with their equipment. The nature of the unit is enumerated before the campaign begins, by assigning the unit a unit type. The unit type assigns a variety of attributes that dictate how it behaves on the battlefield.

Distances on the battlefield are given in this manual and in the program itself as dimensionless integers. In the Second and Third Editions, most Kriegsspiel games were played on battlefields measuring about one meter wide and two meters long, so the distances could be directly reckoned as centimeters. In truth, these distances can be converted to anything, provided the ratios are preserved.

## Unit Attributes

### Team

Every unit belongs to a team, and a given unit’s allegiance is largely, though not entirely, immutable. Ownership can be changed only for naval units via the commandeering of a ship.

### Unit Type

Every unit has a “type.” The standard unit types are universal (see the heading “Units and Unit Types”) but are altered in reality depending on the time and place of the campaign. For instance, the standard unit type “heavy cavalry” would be “hussars” for the Battle of the Brandywine River. These unit types are very important, as attributes are assigned by *type*, rather than by individual unit, which would be far too clunky and slow.

### Health

A unit can represent any number of men, depending on the size and scale of the battle. It can represent as few as a handful of men, or an entire division. Most commonly, a unit represents a battalion of 500 to 1,000 men, but this is by no means standard. Feel free to create campaigns with any desired unit size. Since unit sizes are variable, the health of a unit is measured in Health Points, or HP. When a unit’s Health Points reach zero, it is considered dead and removed from the gamespace.

### Size

In each campaign, a default unit size is set with the value 1. For instance, if every unit begins a campaign as a battalion, a battalion is defined as unit size 1. Therefore, later on, if two units are merged, its unit size is 2. This size value is used to reckon the multiplier for various commands.

### Command Tables

Every command has a command table, which uses the Python dictionary datatype. These command tables have various unit types as keys, while the values are the maximum value for the relevant command for that unit type.

## Communication

Every turn, the player or players representing a particular military will submit their commands to the umpire in writing. All commands for a given turn are to be submitted at once, in the order they should be submitted to their units. Each command should be sent in accordance with its required syntax. Please note though, that the syntax for a player submitting a command may be different from the syntax for submitting a command to Umpire.

# Installing Umpire

## Prerequisites

There are very few computational prerequisites for the installation of the Umpire program itself. All that is required is the newest version of Python 3. If you want to create your own campaigns, it is recommended that you install an integrated development environment, like VisualStudio Code, PyCharm, or Neovim, if you have not already. The writing style of Umpire is such that it can run, without modification, on any platform.

## Downloading the Repository

To download the Umpire program, go to github.com/gtfmadrigal/umpire/releases and download the latest .zip or .tar.gz. Then, unzip or un-tar the archive or tar-ball. Inside the umpire/ directory are the following files and subdirectories:

umpire.py contains the actual code for the Umpire program itself.

README.md contains the GitHub readme file.

LICENSE is the GNU General Public License version 3 (Umpire is an open-source program).

gamefiles/ contains various campaign gamefiles provided by the program.

documentation/ contains all of the documentation, including this manual, a changelog detailing every Git commit, and notes for every full release.

## Creating a Gamefile

### Creating the File Itself

Every Kriegsspiel campaign has a unique gamefile, which establishes the gamespace and units. While some gamefiles are provided for the user in the Umpire repository itself, eventually you will want to create your own.

To start up a new campaign, create a file in the umpire/ directory named after the battle itself, with the .py filename extension. In umpire.py, change line 3 to from x import \* where x is the name of the gamefile. Move any other gamefiles in the umpire/ directory to gamefiles/ so as to not cause confusion. Then, follow along with this guide to create the new gamefile.

Before proceeding, decide on the following:

* The names of the two militaries fighting the battle. Make sure these names are short and easily recognizable (e.g., “Allied” and “Axis,” “Russian” and “French,” etc.)
* All unit types present in the campaign. (For a list of all unit types, see the section titled “Units and Unit Types”.)
* Whether or not air combat will be present.
* The names of all units, and what their unit types are.
* Whether or not there will be fog-of-war, and what that fog will be.

### Defining Variables

The following variables have to be set at the top of the gamefile:

String variables:

* firstTeam should be set to the capitalized name of the first team, like Allied or French.
* secondTeam should be set to the capitalized name of the other team.

Integer variables:

* firstHealthTotal, the starting total health for the first team. After firstTeamTable and secondTeamTable are filled out, add the following lines of code to your gamefile, and run it. Set firstHealthTotal and secondHealthTotal to the output.

print(sum(firstTeamTable.values()))

print(sum(secondTeamTable.values()))

* secondHealthTotal is calculated the same as firstHealthTotal.
* fogOfWar is the fog variable. To find out what it should be set to, consider how many commands you want to not reach their units. If, for instance, you want one out of every six commands to fail, set it to 6. If you do not want any fogOfWar, set it to 1.

Boolean variables:

* airTheater is set to True if there are any air units in the campaign. Note: the presence of aircraft carriers and their ability to launch sorties does not count towards the air theater.

### Defining Dictionaries and Lists

* allUnitTypes is a dictionary of strings, where every key is the *local unit types*, and where the values are *universal unit types*. Note: if the local unit type is the same as the universal, a key-value pair is still required.
* unitTable is a dictionary of strings. For every unit in the campaign, give it a unique name and put in this dictionary as a key. The value should be the unit’s *local* unit type. The local types are mapped to universal ones in the preceding dictionary.
* firstTeamTable is also a dictionary of strings. Every unit belonging to the first team is included in this dictionary as a key string. The values mapped to each key is the starting health of each unit. These health values can be the maximum health values according to the health table provided for by the unit type, or something else; this is left up to the campaign-writer.
* secondTeamTable is the same as firstTeamTable, but for the units that belong to the second team.

### loadGame() Function

The purpose of the loadGame() function is to alter any of the tables that are present in the setup phase of the Umpire program itself. In fact, just after the definitions of the relevant variables, lists, and tables in umpire.py, loadGame() is called in order to initialize the campaign. By default, this function has only a pass statement. If the campaign writer wants, a particular universal unit type’s attributes can be altered by writing some del table[key] or table[key] = value statements. If not, leave this function as it is. There is no need to manually map local unit types to universal unit types, this is done by Umpire under the hood.

## Running a Campaign

### Running Umpire

1. Ensure that line 2 of umpire.py will import the correct gamefile.
2. Open up a new instance of your terminal. The manner in which you do this will differ based on your operating system.
3. Navigate to the umpire/ directory.
4. Run umpire.py with Python 3.
5. If the program will not run, ensure that your version of Python is up to date, and you have not made any mistakes in your gamefile.

### Entering Commands

The shell will look as follows:

x ~ y team % \_

Enter your command at this shell. See the section labelled “Commands” to learn each command’s syntax. X represents the round number, and Y the command number.

# Units and Unit Types

## Army Units

### Infantry

There are three types of infantry units: *regulars*, *engineers*, and *mechanized.* Infantry units can hide, spy, merge and split into larger and smaller meta-units, build fortifications, and move and fire in the same turn. All types of infantry units can participate in the general attack phase, with a maximum damage of 4. Regular and engineer infantry have an initial health of 4 HP, while mechanized infantry units have an initial health of 6 HP. Regular and mechanized infantry units can build fortifications of maximum strength 4, while engineers can build fortifications of maximum strength 8. Regular infantry and engineers have a maximum range of 10 per turn, while mechanized infantry have a range of 15.

### Artillery

There are three types of artillery units: *light, medium,* and *heavy.* All artillery units can hide but are revealed if they fire. Their regular attack damage is at most 4, and they cannot move and fire in the same turn. Light artillery units have an initial health of 8, a movement range of 10, and can fire artillery of maximum damage 8 over a range of 30. Medium artillery units have an initial health of 9, a movement range of 7, and artillery damage of maximum 9 over a range of 25. Heavy artillery units have an initial health of 10, a movement range of 5, and artillery damage of maximum 10 over a range of 20. Artillery units can merge or split into larger or smaller units. All artillery units can be converted into infantry.

### Cavalry

There are three types of cavalry units, just like artillery: *light*, *medium,* and *heavy.* All cavalry units can merge and split, and move and fire in the same turn, but cannot hide or spy. Light cavalry units have an initial health of 10 HP, movement of 10, an ordinary combat phase of 6 and shells of 10 over range 30. Medium cavalry units have an initial health of 12 HP, movement of 7, an ordinary combat phase of 8 and shells of 16 over range 25. Heavy cavalry units have an initial health of 16 HP, movement of 5, an ordinary combat phase of 10 and shells of 20 over range 20. All cavalry units can be converted into infantry.

### Special

A certain type of pseudo-infantry, called *special*, also exist. They have health of 20, movement range of 15 and can hide but cannot spy. They can merge or split and have an ordinary combat phase attack of 20. They can build fortifications of maximum strength 6 and can move and fire in the same turn.

## Naval Units

### Light Vessels

Light vessels are seaborne ships that are not generally meant for combat, but rather for transport or littoral combat. There are three types of light vessels: *corvettes, amphibious*, and *patrols.* All light vessels have a movement range of 15, a boarding successfulness of 6, and can move and fire in the same turn. Corvettes and amphibious ships can drop depth charges to fight submarines. Corvettes must alter heading to change direction more than 45 degrees, have a maximum health of 4, and an attack damage of 6. Amphibious ships do not need to alter heading, have a maximum health of 4, and an attack damage of 4. Patrol boats do not need to alter heading, have a maximum health of 2, and an attack damage of 4.

### Heavy Vessels

Heavy vessels are those ships designed for ocean-based, blue-water combat. There are three types of light vessels: *cruisers*, *destroyers,* and *battleships.* All heavy vessels must change heading in order to alter their course by more than 45 degrees in a particular turn, can drop depth charges, and can move and fire in the same turn. Cruisers have a maximum health of 10, a movement of 7, an attack damage of 16, artillery damage of 20, an air defense of 12, and a boarding effectiveness of 8. Destroyers have a maximum health of 8, a movement of 10, an attack damage of 8, artillery damage of 10, an air defense of 12, and a boarding defense of 10. Destroyers additionally have the power to fire missiles with a maximum damage of 8. Battleships, the largest of the heavy vessels apart from aircraft carriers, have a maximum health of 12, a movement of 5, an attack damage of 12, artillery damage of 16, an air defense of 6, and a boarding defense of 8.

### Carriers

Aircraft carriers are weaker than other heavy vessels but make up for this combat weakness with their ability to launch sorties and transport aircraft and units. Carriers have a health of 16, can move 5 units in a given turn provided they do not exceed a heading change of 45 degrees, deal a maximum of 12 ordinary damage, can fire sorties of damage 8, an air defense of 8, can drop depth charges and board other ships at effectiveness 6, and can move and fire in the same turn.

### Subsurface Vessels

Subsurface vessels come in two types: *attack submarines* and *missile submarines.* Both are hidden by default, have only 1 health point maximum, can move 15 units in a given turn without needing to alter heading, no ordinary attack damage, and can fire torpedoes. However, missile submarines have an added power: they can fire missiles of damage 16, the highest of any missile.

## Air and Space Units

### Fighters

Fighters come in two types: *light* and *heavy.* Both types of fighters can fire missiles of maximum damage 6, can move and fire in the same turn, and can kamikaze ships and other units. Light fighters have a health of 4, a range of 30 units, an attack damage of 4, and a kamikaze effectiveness of 6. Heavy fighters have a health of 8, a range of 15 units, an attack damage of 6, and a kamikaze effectiveness of 8.

### Bombers

Bombers come in two types: *regular* and *stealth.* Both bomber classes have an attack damage of maximum 4, can drop bombs, can fire missiles of maximum 8, drop pulses, and move and fire in the same turn. Regular bombers have a maximum health of 12, a range of 15 units, and can drop bombs of damage 8. Stealth bombers have a maximum health of 10, a range of 10 units, can hide, and can drop bombs of damage 8.

### Transport

Transport planes are primarily meant for ferrying land units to and from locations. They have a maximum health of 12, a range of 30 units, an attack damage of 4, can airlift units, and can move and fire in the same turn.

### Reconnaissance

Reconnaissance planes, which are either *regular* or *drones,* are meant for collecting information. Both types have a maximum health of 4, an attack damage of maximum 4, can spy and hide, and move and fire in the same turn. Regular reconnaissance planes have a range of 20 units. Drones have a range of 30 units and can drop bombs of damage 10.

# Commands

## Meta-Functions

update()

changeList()

evaluate()

prompt()

modification()

fog()

## Umpire Functions

### Change a Unit’s Health

health [unit]

The health command displays the health of a particular unit. It can be called for any live unit. It takes one argument: the unit to be changed. Unlike every other command, the unit need not belong to the team whose turn it is. This is because this is an important root-command, rather than a player-command. Root or umpire functions are for the use of the umpire himself to alter parts of the game state in order to correct an error or represent something not hard coded into Umpire itself.

The health command, when called, will display the current health of the passed unit, and then ask for the new health to give to that unit. The prompt will look like this:

x ~ y team health # \_

The use of an octothorpe instead of a percent sign at the end of the shell prompt indicates that the prompt is being called by an umpire function. If the new value entered by the user is less than or equal to 0, kill is then called. Otherwise, health simply updates the value of the unit in the relevant team dictionary to the new value.

### Kill a Unit

kill [unit]

The kill command is similar to the health command. Instead of altering the value of the unit in its respective team’s table, kill simply removes the key-value pair entirely and then updates the total health values for each team. However, to ensure that the command remains within the gamespace for the purposes of the info command, it is added to a list of dead units, and the global unit-unit type dictionary is not altered. The kill command, while callable by the user, is more of an internal meta-function that is called by more ordinary functions when a unit dies.

### Freeze a Unit for a Turn

freeze [unit]

The freeze command disallows a particular command from moving for the remainder of the turn. This does not “use up” the unit for a particular turn. It, like kill, is rarely issued in and of itself, but is issued to a particular unit after a unit is moved, so that it cannot be moved again.

### Disable a Unit

disable [unit]

The disable command is effectively a double-freeze command. First, disable simply calls freeze for the same unit. Then, it appends the unit to the disabled units list. When a turn ends, every unit included in the disabled units list is also frozen, effectively freezing a unit for a second turn.

### Disallow a Unit from Being Issued New Commands

use [unit]

The use command is called by most functions, so that a unit cannot be issued more than one command in a single turn. All it does is append the unit to the used units list, which is then checked for by some functions before proceeding. This list is cleared at the end of each turn.

### General Combat

attack

The attack function is, with missile, fire, and bomb, the primary way of dealing damage. It must be called at the end of a turn, as it will end the current turn. When called, it sets into three phases, the attack phase, the defense phase, and the save phase, called in order.

During the attack phase, the following prompt is given to the user:

x ~ y team attack % \_

At this prompt, the player will enter either a unit from his team that is attacking, “quit” to exit the function entirely, “defend” to proceed to the defense phase, or “help” to display the help menu. If the user enters a unit, the damage it will deal is calculated by calling the evaluate meta-function, and then added to the total attack damage.

During the defense phase, the following prompt is then shown:

x ~ y team defend % \_

At this prompt, the user enters either a unit that is defending, “quit” to exit entirely, or “save” to proceed to the save phase. Again, for each unit that is entered, a defense damage is calculated and added to the total defense. If, at any point, the total defense damage exceeds the total attack, the attack is considered repelled, and the defense phase ends.

During the save phase, no prompt is shown to the user. The net damage is calculated by subtracting the defense damage from the attack damage. Then, the net damage per unit is calculated by dividing the net damage by the number of defending units. This damage is applied to each defending unit, and units are killed if their health goes below zero. Finally, the score is displayed, and the turn is ended.

### Display the Score

score

When called, the score function calls the update meta-function, and calculates both the raw score and a percent score for both teams. These values are then displayed.

### End the Current Turn

turn

This ends the turn, clearing the temporary lists of used or already moved units. The turn number is incremented by one. Any unit present in the disabled units list is frozen for the last time.

### Display Details

details

The details command displays the name and location of every hidden unit, and the score.

### Display the Help Menu

help

The help command displays an organized list of commands by theater, and every local unit type present in the current campaign.

### Quit the Game

quit

The quit command displays the score and quits the game.

### Merge Units Together

merge

The merge function, which is called with the single-word command of the same name, is used to merge smaller units into a larger one, for ease of issuing commands. It consists of three phases: a unit type retrieval phase, a unit retrieval phase, and a save phase. During the unit type retrieval phase, the user is asked for a local unit type. If this local unit type is not present in the gamefile, an error is thrown, and the function exits. Otherwise, the function proceeds to the next phase. During the retrieval phase, the user is presented with the following prompt:

x ~ y team merge # \_

At this prompt, the user can type either “quit” to exit, a unit name that belongs to the current player, or “save” to proceed to the next phase. Information is retrieved about every unit, so if a single unit that will be merged into a larger one is frozen, disabled, or used, the larger ending unit will also be. Only units of the same type can be merged together. Once “save” is entered, the function moves to the save phase, which gives the following prompt to the user:

x ~ y team unified # \_

At this prompt, the user simply enters a new name for the merged unit. If a unit with the new name already exists, an error message is displayed, and the player can try again. Otherwise, the smaller units are removed from the team dictionary, and the new one is added to both the team dictionary and the big dictionary, while maintaining all the attributes (health, hiddenness, etc.). Finally, the merged unit is added to a dictionary of units whose size has been altered, with its value being the number of units merged together. This is done so that when the larger unit is called upon to deal damage, this damage is multiplied, since it is larger.

### Split Units into Subunits

split [unit]

The split command is used to split up a unit into smaller ones, for better maneuverability or to confuse the enemy. The split function itself is divided into a retrieval phase and a save phase. Before anything else, the split function gets the current health of the unit passed to it. Then, the following prompt is displayed to the user:

x ~ y team split # \_

At this prompt, the user can type the name of a new unit to be split from the one passed to the command, “save” to proceed to the save phase, or “quit” to exit without saving. If the unit entered already exists, an error is thrown. Otherwise, the unit entered is added to a temporary list. In the save phase, the new health for each new unit is calculated as the original unit divided by the number of new units. The initial unit is removed from any lists, and the new units added to any needed ones.

### Proceed to the Next Phase

next or save

In any function that has a prompt phase, the save command can be issued to proceed to the save phase or the next phase. During the air phase of a turn, the next command can be entered so as to proceed to the ordinary shell.

## Theater-agnostic Functions

### Move a Unit

move [unit]

The move command does exactly what you might expect: it moves a unit. The unit that is passed to the move function is added to a list of units that have moved that turn, so that they cannot be moved again until the next turn.

### Hide a Unit from the Gamespace

hide [unit]

A unit that is hidden with this command is removed from the gamespace. It is still *there*; it simply does not appear. A prompt is shown to the player, provided the unit passed to the hide command is able to hide, as such:

x ~ y team hide % \_

At this prompt, the umpire enters the location of the hidden unit. This data is added to a string called “secrets,” which is displayed when the user calls the details command.

### Reveal a Hidden Unit

reveal [unit]

The reveal function simply returns a hidden unit to the gamespace, making it visible to the players. Additionally, a statement that the unit is no longer hidden is added to the “secrets” variable.

### Acquire Military Intelligence

spy [unit]

The spy function is used to acquire information about the gamespace that is not immediately visible to the player, such as hidden units and unit strength. Provided that the unit passed is able to spy, an effectiveness is generated as a number between 1 and 6. If the effectiveness is 6, the umpire will pass good information to the player. If the effectiveness is 1, bad information will be passed. Otherwise, the umpire will pass no information.

### Fire Projectiles that Cannot be Defended Against

fire [unit]

The fire function is used to inflict damage *from* one unit *onto* multiple enemy units. This damage cannot be defended against. This function is divided into three sections: the attack phase, the defense phase and the save phase. Unlike attack, only one unit can attack, and that unit is passed to the function, so the attack phase consists only of checks to make sure that the unit is able to fire. During the defense phase, the following prompt is shown to the player:

x ~ y team fire % \_

At this prompt, the player can either enter “quit” to exit, an enemy unit to be fired upon, or “save” to proceed to the save phase. If a valid enemy unit is entered, it is added to a defending units list. During the save phase, the total attack damage is calculated for the attacking unit. The per unit damage is found by dividing this damage by the length of the defending units list. Every unit’s health in the defending units list is reduced by the per-unit damage, and if the damage goes below 0, that unit is killed. This information and the resulting score is displayed to the player.

## Army Functions

### Build a Structure

build [unit]

The build command is used to create a fortification or other structure. Its strength depends on the maximum strength the unit passed to it can build.

### Fire Projectiles that Can be Defended Against

missile [unit]

The attack command is for multi-unit attacks with multi-unit defenses; the fire command is for single-unit attacks with multi-unit damages and no defense. The missile command, by contrast is for single-unit against single-unit attacks, which can be defended against. Using the missile command, along with the fire and attack commands, reveals the unit that uses it. Assuming that the passed unit has the ability to fire missiles, the function is split into a retrieval phase and a save phase. During the retrieval phase, the function determines the maximum possible damage dealt by the attack missile. Then, a prompt is shown to the user:

x ~ y team missile % \_

At this prompt, the user enters the target unit. If the target unit can also fire missiles, its defense is equal to the amount of damage it would deal if it had been issued the missile command; then, the net damage is calculated and dealt, if any. If not, the target unit suffers the total amount of damage. If the target unit’s health goes below zero, it is killed. The score is then displayed.

### Change a Unit’s Type

convert [unit]

The convert command changes the unit type to infantry.

## Naval Functions

### Alter the Heading of a Ship

heading [unit]

Heavy ships cannot alter their heading using the move command by more than 45 degrees in a given turn. If you wish to change the direction a ship changes by more than that, the heading command must be issued. A ship that was issued the heading command cannot be moved in the same turn.

### Torpedo a Ship

torpedo [unit]

The torpedo function can only be issued to submarines against ships. It consists of two phases: retrieval and save. During the retrieval phase, a random number between 1 and 6 is generated, and the following prompt is displayed to the player:

x ~ y team torpedo % \_

At this prompt, the user enters the target ship. If the random number was 6, the ship is sunk. Otherwise, between 1 and 5 damage is inflicted to the ship, and if the damage causes the ship’s health to reach 0, it is also sunk. The score is then displayed.

### Launch Air Sorties

sortie [unit]

Aircraft carriers can launch sorties of bombers against enemy ships. Note: this air attack is separate from the air phase. Like many other functions, it consists of a retrieval phase and a save phase. First, the function gets the maximum damage possibly inflicted. Then, the user enters the target ship at the following prompt:

x ~ y team sortie % \_

The target ship’s defense is then calculated, and the resulting damage inflicted upon it.

### Drop Depth Charges

depthcharge [unit]

To defend against submarines, ships can drop depth charges. The target submarine is retrieved from the player at this prompt:

x ~ y team depthcharge % \_

A random effectiveness from 1 to 6 is generated. If the effectiveness is 6, the submarine is sunk. If the effectiveness is 5, the submarine is disabled. Otherwise, nothing happens and the function returns.

### Commandeer a Ship

board [unit]

A ship can attempt to commandeer, that is, steal, another ship from its original owner, using the board command. First, the target ship is acquired from the player with this prompt:

x ~ y team board % \_

A random effectiveness from 1 to 6 is generated. If the effectiveness is 5 or 6, the ship is seized and its ownership changes. Otherwise, the boarding attempt false, but some damage is inflicted on the target ship. However, since the ship attempting to board is disabled, this leaves it susceptible to a great deal of damage during the next turn.

## Air/Space Warfare Functions

### Take Off a Plane

takeoff [unit]

As a prerequisite to all other functions in the air shell, a plane must first be airborne. If a plane is not airborne, it cannot use any be issued any other command.

### Land a Plane

land [unit]

At the end of a turn, any plane that is still in the air will fall out of the sky and be destroyed. To prevent this, land every plane.

### Drop an Electromagnetic Pulse

pulse [unit]

A pulse command disables units but does not deal any damage. First, an effectiveness is calculated. If the effectiveness is 6, the function proceeds to the pulse phase itself. Otherwise, the pulse is considered ineffective, and the function quits. At the start of the pulse phase, the following prompt is shown:

x ~ y team pulse % \_

At this prompt, any number of units can be entered, or “quit” to quit without saving, or “save” to save. At the save phase, the pulsed units are disabled.

### Airlift Units

airlift [unit]

An infantry unit can be transported by airlift via transport planes. The unit to be transported is done so via the following prompt:

x ~ y team airlift % \_

### Fly a Plane into an Army or Naval Unit

kamikaze [unit]

Fighter planes can fly directly into a ship or another unit to damage or destroy it, though this also destroys the fighter. A target unit is acquired via the following prompt:

x ~ y team kamikaze % \_

An effectiveness from 1 to 6 is generated, and the health of the unit is retrieved. If the effectiveness is 6, or the effectiveness minus the current health of the unit is less than or equal to 0, the target unit is killed. Otherwise, the effectiveness value is dealt to the unit as damage.

### Drop Bombs

bomb [unit]

The bomb function can be issued to bombers. It functions much like the fire command, with one damage source and multiple targets that cannot defend. However, the prompt looks like this:

x ~ y team bomb % \_