**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 universal unit type.
* 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 ~ x firstTeam %

Enter your command at this shell. See the section labelled “Commands” to learn each command’s syntax.

# Units and Unit Types

## Army Units

### Infantry

### Artillery

### Cavalry

### Special

## Naval Units

### Light Vessels

### Heavy Vessels

### Carriers

### Subsurface Vessels

## Air and Space Units

### Fighters

### Bombers

### Transport

### Reconnaissance

# Commands

## Umpire Functions

### Change a Unit’s Health

### Kill a Unit

### Freeze a Unit for a Turn

### Change a Unit’s Type

### Disable a Unit

### Disallow a Unit from Being Issued New Commands

### General Combat

### Display the Score

### End the Current Turn

### Display Details

### Display the Help Menu

### Quit the Game

### Merge Units Together

### Split Units into Subunits

### Proceed to the Next Phase

## Theater-agnostic Functions

### Move a Unit

### Hide a Unit from the Gamespace

### Reveal a Hidden Unit

### Acquire Military Intelligence

### Fire Projectiles that Cannot be Defended Against

## Army Functions

### Build a Structure

### Fire Projectiles that Can be Defended Against

## Naval Functions

### Alter the Heading of a Ship

### Torpedo a Ship

### Launch Air Sorties

### Drop Depth Charges

### Commandeer a Ship

## Air/Space Warfare Functions

### Take Off a Plane

### Land a Plane

### Drop an Electromagnetic Pulse

### Airlift Units

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

### Drop Bombs

# Documentation

## Definitions

### Imports

### Variables

### Lists

### Dictionaries

## Meta-Functions

### update

### changeList

### evaluate

### prompt

### modification

### fog

## One Word Commands

### score

### turn

### details

### quitGame

### helpText

### attack

## Umpire Functions

### health

### kill

### freeze

### convert

### disable

### use

### merge

### split

## Theater-Agnostic Functions

### move

### hide

### reveal

### spy

### fire

## Naval Functions

### heading

### torpedo

### sortie

### depthcharge

### board

## Army Functions

### build

### missile

## Air Functions

### takeoff

### land

### pulse

### airlift

### kamikaze

## Shell

### info

### umpireShell

### airShell

### shell

### Game Loop

# Glossary