The LaTeX wargame package

A tutorial

Christian Holm Christensen

November 19, 2024

1 Introduction

This is a short tutorial on how to use the LaTeX wargame package. We will walk through the definition materials for a game. We will *not* give actual rules for the game. That is a whole different topic and I refer you to the literature, for example

Simulating war by P.Sabin

The Complete Wargames Handbook by J.F.Dunnigan

Designing Wargames: Introduction by G.Phillies (and associated Youtube lecture series)

2 The game

The game we will create is a small game with two factions (sides, or players). The game is *not* supposed to be play-able, and we'll leave out a lot of details which we would need in a full game.

3 The game package

To make our game components re-usable (in particular when we want to make a VASSAL module), we will put our definitions of counters, board, and charts into a package file called `game.sty`[1]

Code shown below is in the package.

The first thing we do in the package is to identify the package and load the wargame package.

```
\ProvidesPackage{game}
\RequirePackage{wargame}
\RequirePackage{colortbl}
```

4 The units

We will set-up our units for the game. As noted above, we will make two factions which we will call *A* and *B*.

Figure 1: Faction colours

The NATO App6(d)[2] symbology defines that friendly and hostile units should have different symbol frames, but as we will make a game for two players, and it seems unfair to label one as *hostile* and the other *friendly*, we will stick with the `friendly` base frames.

Of course, if we were to make a solitaire or cooperative game, we might want to use the `hostile` base frame for the opponent units.

4.1 Faction styles

We will start by defining two Ti*k*Z/PGF[3] styles for our two factions. We will call these `a` and `b` (obviously). These will define the colours of all counters of those sides.

```
\colorlet{a-bg}{hostile}
\colorlet{b-bg}{friendly}
\tikzset{%
  a/.style={fill=a-bg,draw=black},
  b/.style={fill=b-bg,draw=black}}
```

Note that we made the colours `a-bg` and `b-bg`. Since we will use these colours a few times, it makes sense to make a single definition which we can then freely change at any point and then automatically have that change propagate everywhere.

These styles are shown in Figure 1.

Here we have used the colours `friendly` and `hostile` defined by the wargame package, even though we said we would not use the corresponding `hostile` base frame.

[1] In fact, most of this document is in that file, because it allows us to document the code using LaTeX's `ltxdoc` class.

[2] https://nso.nato.int/nso/nsdd/main/standards/ap-details/1912/EN

[3] The wargame package relies heavily on Ti*k*Z/PGF. It is highly recommended that you acquaint yourself with Ti*k*Z/PGF. The manual, including tutorials, is available from https://ctan.org/pkg/pgf.

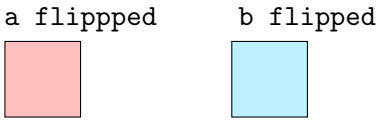


Figure 2: Faction flipped colours

However, we can use any colours we can define in LaTeX with the xcolor[4] package, for example,

```
\definecolor{a-bg}{HTML}{3333ff}
```

which will make a light blue colour.

Since we intent to make *double-sided* counters, so that units will have two steps (front and back), we also define styles for the back side. These will have lighter backgrounds than the front side.

```
\tikzset{%
  a flipped/.style={a,fill=pgffillcolor!50!white},
  b flipped/.style={b,fill=pgffillcolor!50!white}}
```

Note that we use the styles `a` and `b` as bases for these styles. Thus, if we make changes to the base styles, they automatically propagate to our flipped styles.

> By convention, it is best to name your backsides of counters and so on the same as the front side, but with ‘ `flipped`’ appended. It is therefore also a good idea to use that convention for styles and such.

The colour `pgffillcolor` is what ever the current fill colour is (i.e., set by the `a` and `b` styles).

These styles are shown in Figure 2.

4.2 Unit templates

In our game we will have two kinds of units: land, and air.

Land units represent ground types of different kinds (infantry, armoured, etc.) and have two *factors*: A combat factor (CF) and a movement factor (MF). In addition, for artillery units, we will also specify a range. Each unit will also have a unique identifier, and possibly parent organisational identifier, starting hex, and turn of appearance.

Air units provide different kinds of support for the ground units. Since we will make a game on an *operational* level (armies, divisions, brigades), we will give air units a single factor — the odds column shift (see the combat resolution table later on).

[4] https://ctan.org/pkg/xcolor

Let us first define a template for the ground units. We will call this `gu` (for ground unit), and it will take 8[5] arguments:

1. The unit type
2. The lower unit type (e.g., airborne)
3. The unit size (echelon)
4. The unit identifier
5. The parent unit identifier
6. The factors (more on this later)
7. The starting hex
8. The turn of appearance

```
\tikzset{
  gu/.style args={#1,#2,#3,#4,#5,#6,#7,#8}{%
    chit={%
      symbol={% Defines the NATO symbol
        faction=friendly, % See note
        command=land, % Ground units
        main={#1},% Unit type(s) (e.g., infantry)
        lower=#2,% Lower type
        echelon=#3,% Size (e.g., division)
        scale line widths,
        line width=1pt,
      },
      unique={chit/small identifier=#4}, % Unit ID
      parent={chit/small identifier=#5}, % Parent ID
      factors={#6}, % The unit factors
      upper left={chit/small identifier=#7}, % Hex
      upper right={gu turn=#8} % Turn
    } % end of chit
  } % end of gu
}
```

Note the `unique`, `parent`, and `upper left` keys of `chit` are set to contain a `chit/identifier` (or `chit/small identifier`) picture. In general, all the keys of a `chit` and (and `natoapp6c`) Ti*k*Z/PGF node need to be `pic`[6] objects. The `chit/identifier` picture outputs the text (the argument after the `=`).

Note that for `upper left`, we used the `pic gu turn`, which we have not defined yet. Let us define that now. This will be a picture that puts in the turn number when a unit appears. Turn number “0” is the “At-start” turn, and so we will deal with that specifically. Other turns should just be done normally — that is, we use another `chit/small identifier` picture

```
\tikzset{
  gu turn/.pic={%
    \ifx#1\empty\else
      \ifnum0=#1\else%
      \pic{chit/small identifier={#1}};\fi\fi},
  pics/gu turn/.default=0
}
```

[5] Yes, that’s a lot, but we may leave some of them blank. Don’t worry, we’ll make more short-hands.

[6] A small re-usable picture. See the Ti*k*Z/PGF manual, Chapter 18.

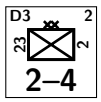


Figure 3: Ground unit template

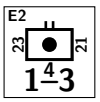


Figure 4: Combat and field artillery templates

Note that we set the default value in case we get no turn number.

We have put the keys `scale line widths` and `line width=1pt` into the `symbol` (the NATO symbol) part. The latter sets the line width to be a little thicker than usual. This is to make the counters more easily read. The former ensures that if we scale our counters up or done, then the line width will likewise scale[7].

For the factors, we will use two different `pic`: `chit/2 factors` for regular ground units, and `chit/2 factors artillery` for artillery units.

The template is shown in Figure 3.

The image above was made with

```
\begin{tikzpicture}
  \chit[gu={infantry,,division,23,2,
            {chit/2 factors={2,4}},D3,2}];
\end{tikzpicture}
```

As said above, we will have artillery ground units and other kinds of ground units. Let us make two templates — one for field-artillery ground units (`fu`), which has a range, and regular combat units (`cu`).

First the artillery unit type: `fu`. This takes 8 arguments

1. The echelon
2. The identifier
3. The parent identifier
4. The combat factor (CF)
5. The movement factor (MF)
6. The range
7. The starting hex
8. The turn of appearance

```
\tikzset{
  fu/.style args={#1,#2,#3,#4,#5,#6,#7,#8}{
    gu={%
      {{[fill=pgfstrokecolor]artillery}}, % Type
      ,%lower
      #1, % echelon
      #2, % ID
      #3, % Parent ID
      {chit/2 factors artillery={#4,#5,#6}},
      #7, % Hex
      #8  % Turn
    }
  }
}
```

Note that we pass the options `[fill=pgfstrokecolor]` to the `pic artillery`[8] to fill in the artillery symbol. Note that we need to protect this by putting in curly brackets (2 in this case). Similarly, we protect the `chit/2 factors artillery` call.

> In general, we may pass options (or keys) to a `pic` by preceeding its name with `[`⟨*keys*⟩`]`. However, we then *must* protect the it as in {`[`⟨*keys*⟩`]`⟨*pic*⟩}. Further, if `[`⟨*keys*⟩`]` contains a list of keys, separated by commas, then we need to protect ⟨*keys*⟩ too, as in {`[{[`⟨*keys*⟩`]}]`⟨*pic*⟩}. A little clunky, but that's how TEX works sometimes.

And now the combat unit template: `cu`. This takes 9 arguments.

1. The type
2. The lower type
3. The echelon
4. The identifier
5. The parent identifier
6. The combat factor (CF)
7. The movement factor (MF)
8. The starting hex
9. The turn of appearance

```
\tikzset{
  cu/.style args={#1,#2,#3,#4,#5,#6,#7,#8,#9}{
    gu={%
      {#1}, % Type
      #2, % lower
      #3, % echelon
      #4, % ID
      #5, % Parent
      {chit/2 factors={#6,#7}},
      #8, % Hex
      #9  % Turn
    }
  }
}
```

Again, let us see what that looks like (Figure 4).

[7] Other elements scale automatically because of Ti*k*Z/PGF's `transform shape` key

[8] To be explicit `natoapp6c/s/artillery`.

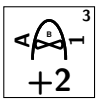


Figure 5: Air unit template

The above was made with

```
\begin{tikzpicture}
  \chit[fu={battalion,23,21,1,3,4,E2,}](0,0);
  \chit[cu={infantry,airborne,regiment,21,2,6,4,E2,}]
  (2,0);
\end{tikzpicture}
```

Finally, we make a template for air units. We will have Close-combat Air Support (CAS `cas`) and strategic bombers (`sb`). To make things a little easier for us, we first define a template for all air units. This takes 7 arguments:

1. The unit type (e.g., `fixed wing`, `rotary wing`).
2. The upper modifier (e.g., `F` for fighter)
3. The lower modifier (e.g., `H` for 'heavy')
4. The unit identifier
5. The parent unit identifier
6. The factor (column shift)
7. The turn of appearance

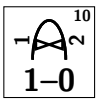
```
\tikzset{
  au/.style args={#1,#2,#3,#4,#5,#6,#7}{%
    chit={%
      symbol={% Defines the NATO symbol
        faction=friendly, % See note
        command=air, % Air units
        main=#1,% Unit type
        upper={text=#2},
        lower={text=#3},
        scale line widths,
        line width=1pt,
      },
      unique={chit/identifier=#4}, % Unit ID
      parent={chit/identifier=#5}, % Parent ID
      factors={#6}, % The unit factors
      upper right={gu turn=#7} % Turn
    } % end of chit
  } % end of gu
}
```

Here, `text`[9] for `lower` and `upper` is a picture that puts the text given as the argument after the `=`. The template is shown in Figure 5.

The above was made with

```
\begin{tikzpicture}
  \chit[au={fixed wing,B,,A,1,
            {chit/1 factor=+2},3}](0,0);
\end{tikzpicture}
```

[9] really `/natoapp6c/s/text`.



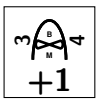


Figure 6: CAS and SB templates

As we said, we will have CAS and strategic bomber wings in this game, so we make templates for these. Our CAS will be helicopters (`rotery wing`) while the strategic bombers are planes. CAS gives a 1 CF, while strategic bombers gives 1 column shift. Arguments are

1. The identifier
2. The parent identifier
3. The turn of appearance

```
\tikzset{
  cas/.style args={#1,#2,#3}{ % Close air support
    au={% Air unit
      rotary wing, % Helicopter
      ,, % No lower, upper
      #1, % ID
      #2, % Parent ID
      {chit/2 factors={1,0}},
      #3 %Turn
    }
  },
  sb/.style args={#1,#2,#3}{% strategic bomber
    au={% Air unit
      fixed wing, %Planes
      B, % Bomber
      M, % Medium
      #1, % ID
      #2, % Parent
      {chit/1 factor={+1}},
      #3 % Turn
    }
  }
}
```

The templates are shown in Figure 6.

The above was made with

```
\begin{tikzpicture}
  \chit[cas={1,2,10}(0,0);
  \chit[sb={3,4,}(2,0);
\end{tikzpicture}
```

4.3 Specialisations

We will now make some specialisations of the `gu` style. These represent headquarters `hq`, mechanised infantry `mi`, infantry `in`, and so on. We do this because we want units of the same kind to have similar factors and so on. Since almost all of our units are battalions, we also code that in.

All of these, except `ff` for field artillery units, take 4 arguments

1. The identifier
2. The parent identifier
3. The starting hex
4. The turn of appearance

The style `ff` precedes these arguments with a single argument for the unit size (echelon).

```
\tikzset{
  in/.style args={#1,#2,#3,#4}{%
    cu={infantry,,battalion,#1,#2,2,3,#3,#4}},
  mi/.style args={#1,#2,#3,#4}{%
    cu={{armoured,infantry},,battalion,#1,#2,4,4,#3,#4}},
  ab/.style args={#1,#2,#3,#4}{%
    cu={infantry,airborne,battalion,#1,#2,1,3,#3,#4}},
  ca/.style args={#1,#2,#3,#4}{%
    cu={combined arms,,battalion,#1,#2,3,3,#3,#4}},
  ar/.style args={#1,#2,#3,#4}{%
    cu={armoured,,battalion,#1,#2,6,4,#3,#4}},
  re/.style args={#1,#2,#3,#4}{%
    cu={reconnaissance,,battalion,#1,#2,6,5,#3,#4}},
  ff/.style args={#1,#2,#3,#4,#5}{%
    fu={#1,#2,#3,2,5,3,#4,#5}},
  hq/.style args={#1,#2,#3,#4,#5}{
    cu={headquarters,,#1,#2,#3,0,1,#4,#5}},%
  hqbg/.style args={#1,#2,#3,#4}{
    hq={brigade,#1,#2,#3,#4}},
  hqregt/.style args={#1,#2,#3,#4}{
    hq={regiment,#1,#2,#3,#4}},
}
```

4.4 The actual units

Above we defined templates for the various units. These templates saves us a lot of trouble when defining the *actual* units. Below we will make the units for each faction in the game. We will define them as Ti*k*Z/PGF styles, just like we did for the templates.

> Above we said we want to make double sided counters. We will get back to the back-side (flipped) versions of the counters in a moment.

4.4.1 Side A

Below we will define the units without much commentary. Note that we use the style `a` for all our units so that they get the right style. Note that we pass the single argument `#1` as the turn of appearance to all units. This will be used when generating an *Order of Battle* later on.

```
\tikzset{
  a hq/.style       ={a,hq={corps,,,A2,#1}},
  a 1 hqbg/.style   ={a,hqbg={1,,A2,#1}},
  a 1 1lg ibn/.style={a,mi={I-LG,1,D3,#1}},
  a 1 1gh ibn/.style={a,mi={I-GH,1,C2,#1}},
  a 1 2jd ibn/.style={a,mi={II-JD,1,A2,#1}},
```

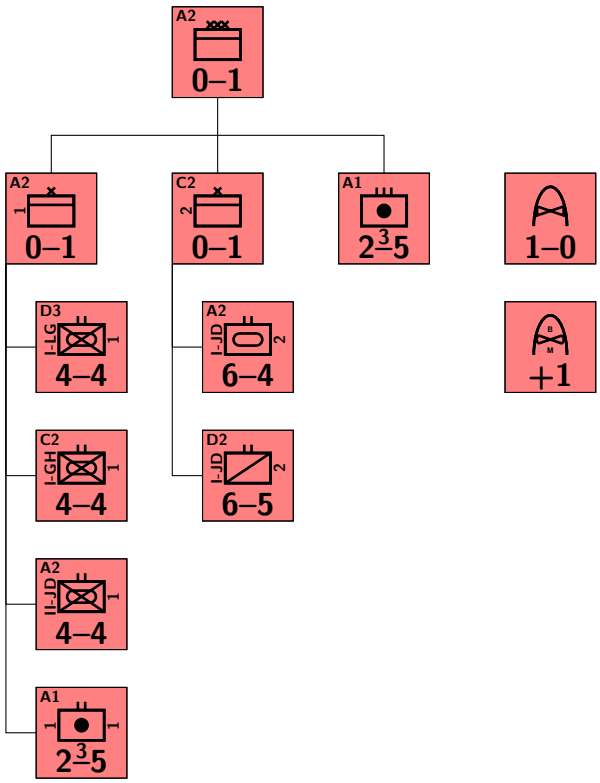


Figure 7: Faction A organisational chart

```
  a 1 1 abn/.style  ={a,ff={battalion,1,1,A1,#1}},
  a 2 hqbg/.style   ={a,hqbg={2,,C2,#1}},%
  a 2 1jd abn/.style={a,ar={I-JD,2,A2,#1}},%
  a 2 3gh rbn/.style={a,re={I-JD,2,D2,#1}},%
  a aregt/.style={a,ff={regiment,,,A1,#1}},%
  a f/.style={a,cas={,,#1}},%VR
  a b/.style={a,sb={,,#1}}%SK
}
```

Let us draw these units in what looks like an organisational diagram (Figure 7).

Before we go on to side B, we will make a macro that contains a list of all our A side counters.

```
\def\alla{{%
    a hq,
    a 1 hqbg,
    a 1 1lg ibn,
    a 1 1gh ibn,
    a 1 2jd ibn,
    a 1 1 abn,
    a 2 hqbg,
    a 2 1jd abn,
    a 2 3gh rbn,
    a aregt,
    a f,
    a b}}
```

4.4.2 Side B

Below we will define the units without much commentary. Note that we use the style `b` for all our units so that they get the right style. Note that we pass the single argument `#1` as the turn of appearance to all units.

This will be used when generating an *Order of Battle* later on. ‘

```
\tikzset{
  b hq/.style=          {b,hq=    {corps,,,F6,#1}},
  b lg hqregt/.style=   {b,hqregt={LG,,F6,#1}},
  b lg ibn/.style=      {b,in=    {LG,,F6,#1}},
  b k3 hqregt/.style=   {b,hqregt={K3,,E7,#1}},
  b k3 31 abibn/.style= {b,ab=    {31,K3,E7,#1}},
  b k4 hqregt/.style=   {b,hqregt={K4,,F10,#1}},
  b k4 abibn/.style=    {b,ab=    {NL,K4,F10,#1}},
  b p4 hqregt/.style=   {b,hqregt={P4,,D7,#1}},
  b p4 41 cabn/.style=  {b,ca=    {41,P4,D7,#1}},
  b p4 42 cabn/.style=  {b,ca=    {42,P4,D7,#1}},
  b p7 hqregt/.style=   {b,hqregt={P7,,D4,#1}},
  b p7 71 ibn/.style=   {b,mi=    {71,P7,D4,#1}},
  b p7 72 cabn/.style=  {b,ca=    {72,P7,D4,#1}},
  b i13 hqregt/.style=  {b,hqregt={I13,,D8,#1}},
  b i13 131 ibn/.style= {b,in=    {131,I13,D8,#1}},
  b i13 132 ibn/.style= {b,in=    {132,I13,D8,#1}},
  b p18 hqregt/.style=  {b,hqregt={P18,,F5,#1}},
  b p18 181 cabn/.style={b,ca=    {181,P18,F5,#1}},
  b i19 hqregt/.style=  {b,hqregt={I19,,F10,#1}},
  b i19 191 cabn/.style={b,ca=    {191,I19,F10,#1}},
  b i19 192 cabn/.style={b,ca=    {192,I19,F10,#1}},
  b i21 hqregt/.style=  {b,hqregt={I21,,D10,#1}},
  b i21 211 ibn/.style= {b,in=    {211,I21,D10,#1}},
  b i21 212 ibn/.style= {b,in=    {212,I21,D10,#1}},
  b f/.style=           {b,cas=   {F17,BL,#1}},
  b b/.style=           {b,sb=    {F7,SA,#1}}
}
```

Let us draw these units in what looks like an organisational diagram (Figure 8).

As before, we will make a macro that contains all B counters.

```
\def\allb{{
    b hq, b lg hqregt, b lg ibn,
    b k3 hqregt, b k3 31 abibn,
    b p4 hqregt, b p4 41 cabn, b p4 42 cabn,
    b p7 hqregt, b p7 71 ibn,  b p7 72 cabn,
    b i13 hqregt, b i13 131 ibn,b i13 132 ibn,%
    b f, b b%
  },{},{},{% Empty turns
    b p18 hqregt, b p18 181 cabn%
  },{
    b i21 hqregt, b i21 211 ibn, b i21 212 ibn%
  },{},{
    b k4 hqregt, b k4 abibn%
  },{
    b i19 hqregt, b i19 191 cabn, b i19 192 cabn%
  }}
```

Note that this is defined as a list of lists, and some of the lists are a little special. This is because we will reuse this list over and over again, and in particular for the Order of Battle charts. The point is that each element of the outer most list correspond to a turn, starting with turn “0”, or “At start”. Empty elements are thus turns where there will be no reinforcements for the faction.

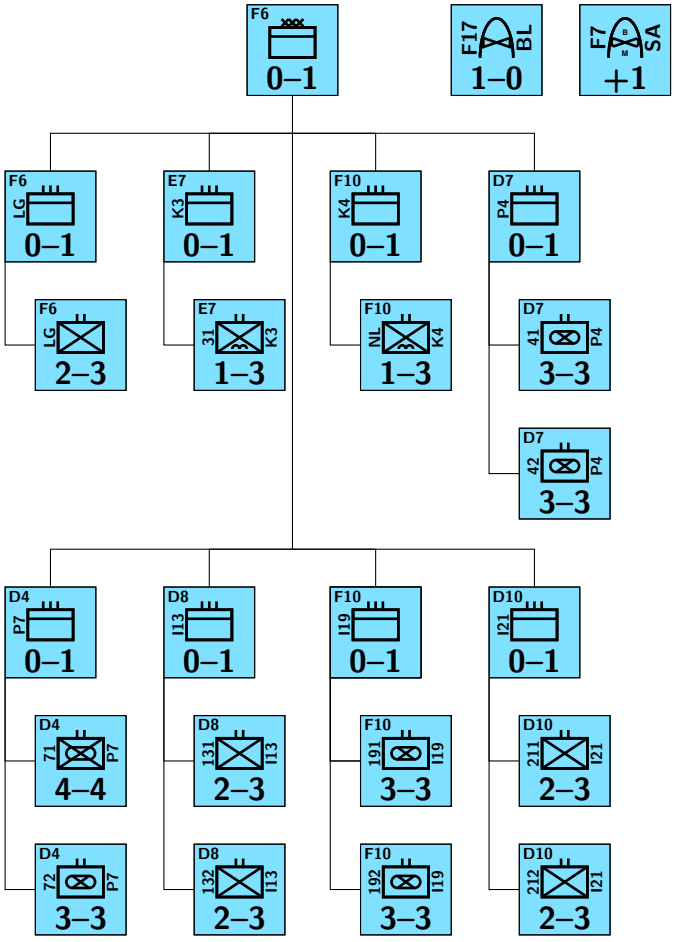


Figure 8: Faction B organisational chart

4.5 The `flipped` side of things

Well, really the `flipped` definitions of the units. The flipped side of our unit counters will represent a "spent" state, i.e., a state where the unit cannot attack and only do limited manoeuvres. In this state, all units will have a combat factor or zero and movement factor of 1. Thus, we define a simple style that sets the factors

```
\tikzset{
  flipped/.style={%
    /chit/factors={chit/2 factors={0,1}},
    /chit/upper left={},
    /chit/upper right={}
  }
}
```

Note that we give the *explicit full* path to the chits factors key. This is because we want to *monkey patch* our units to have these new factors. We also clear the upper left and right corners on the flip side.

Now, we'll do a little hack. Previously, we defined the `a flipped` and `b flipped` styles, but if you look closely, we can see how we can automatically define the flipped fill colours. Thus we make another style that does this.

```
\tikzset{%
  f/.style={%
    flipped,fill=pgffillcolor!50!white}}
```

With this, we can define the `flipped` state of all A counters.

```
\tikzset{%
  a hq flipped/.style=  {a hq       ,f},
  a 1 hqbg flipped/.style= {a 1 hqbg   ,f},
  a 1 1lg ibn flipped/.style= {a 1 1lg ibn,f},
  a 1 1gh ibn flipped/.style= {a 1 1gh ibn,f},
  a 1 2jd ibn flipped/.style= {a 1 2jd ibn,f},
  a 1 1 abn flipped/.style= {a 1 1 abn  ,f},
  a 2 hqbg flipped/.style= {a 2 hqbg   ,f},
  a 2 1jd abn flipped/.style= {a 2 1jd abn,f},
  a 2 3gh rbn flipped/.style= {a 2 3gh rbn,f},
  a aregt flipped/.style= {a aregt    ,f},
  a f flipped/.style=  {a,f},
  a b flipped/.style=  {a,f},
}
```

Let us draw the A side counters, front and back, using the macro `doublechits` (Figure 9).

The above was produced by

```
\begin{tikzpicture}
  \doublechits{\alla}{3}{0.04}
\end{tikzpicture}
```

We will use the macro `\doublechits` later on to make our chit sheets. This works *only* because we define the back sides of our chits to have the style '⟨*front*⟩ `flipped`', which is why it is *highly* recommended to follow this convention.

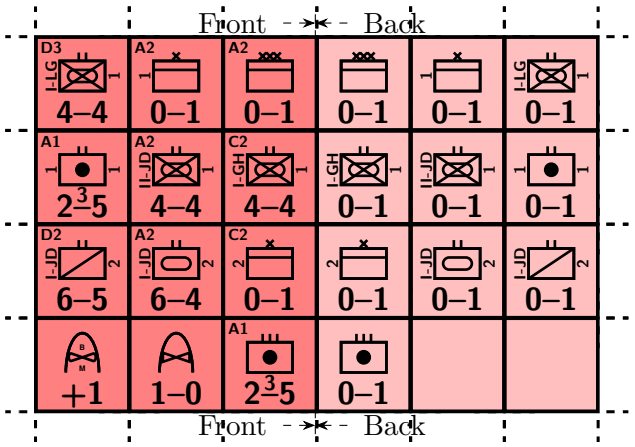


Figure 9: All faction A counters, front and back

We do the same thing for the faction B counters.

```
\tikzset{%
  b hq flipped/.style=  {b hq,          f},
  b lg hqregt flipped/.style= {b lg hqregt,   f},
  b lg ibn flipped/.style= {b lg ibn,      f},
  b k3 hqregt flipped/.style= {b k3 hqregt,   f},
  b k3 31 abibn flipped/.style= {b k3 31 abibn, f},
  b k4 hqregt flipped/.style= {b k4 hqregt,   f},
  b k4 abibn flipped/.style= {b k4 abibn,    f},
  b p4 hqregt flipped/.style= {b p4 hqregt,   f},
  b p4 41 cabn flipped/.style= {b p4 41 cabn,  f},
  b p4 42 cabn flipped/.style= {b p4 42 cabn,  f},
  b p7 hqregt flipped/.style= {b p7 hqregt,   f},
  b p7 71 ibn flipped/.style= {b p7 71 ibn,   f},
  b p7 72 cabn flipped/.style= {b p7 72 cabn,  f},
  b i13 hqregt flipped/.style= {b i13 hqregt,  f},
  b i13 131 ibn flipped/.style= {b i13 131 ibn, f},
  b i13 132 ibn flipped/.style= {b i13 132 ibn, f},
  b p18 hqregt flipped/.style= {b p18 hqregt,  f},
  b p18 181 cabn flipped/.style={b p18 181 cabn,f},
  b i19 hqregt flipped/.style= {b i19 hqregt,  f},
  b i19 191 cabn flipped/.style={b i19 191 cabn,f},
  b i19 192 cabn flipped/.style={b i19 192 cabn,f},
  b i21 hqregt flipped/.style= {b i21 hqregt,  f},
  b i21 211 ibn flipped/.style= {b i21 211 ibn, f},
  b i21 212 ibn flipped/.style= {b i21 212 ibn, f},
  b f flipped/.style=  {b,f},
  b b flipped/.style=  {b,f}}
```

Let us draw the faction B counters, front and back, using the macro `doublechits` (Figure 10).

The above was produced by

```
\begin{tikzpicture}
  \doublechits{\allb}{3}{.04}
\end{tikzpicture}
```

4.6 Other counters

In principle one can make as many and different counters and markers as needed. Here, we will keep things simple and only define a game turn marker. The package wargame already defines one as shown in Figure 11.

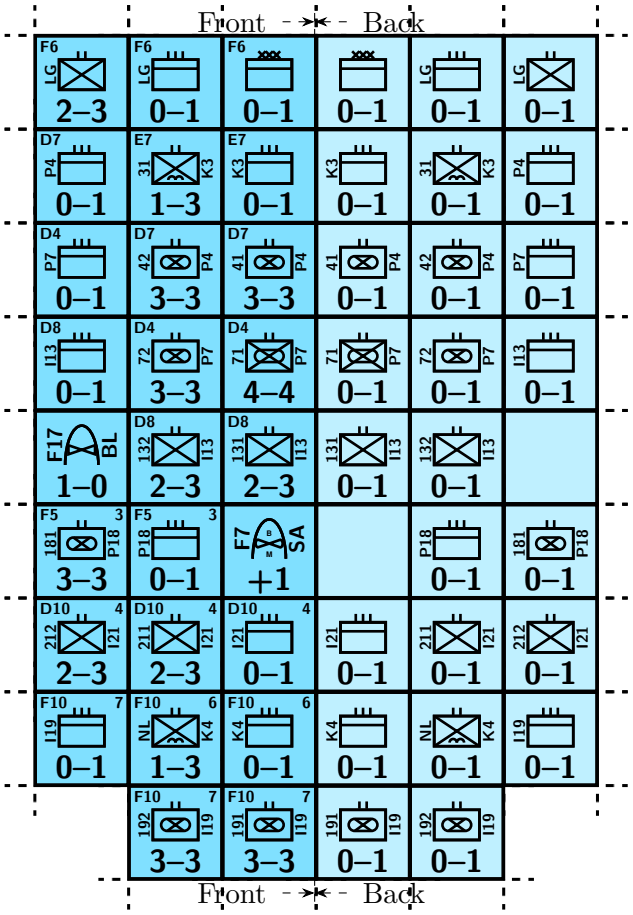


Figure 10: All faction B counters, front and back

Figure 11: The wargame game turn counter

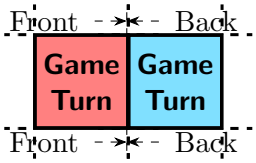


Figure 12: Modified game turn, front and back

However, this has no flip side and we would like the side to reflect the faction currently in turn

```
\tikzset{
  game turn chit/.append style={a},
  game turn chit flipped/.append style={b}
}
```

This modified game turn is shown in Figure 12.

5 The board

Designing the board is probably where one will spend the most time. The wargame package is not omnipotent, but tries to make it as simple as possible. However, some artistic streak is a good thing, and familiarity with Ti*k*Z/PGF is highly recommended.

The simplest thing one can do is to import an image and superimpose hexes on top of that image. While often a good solution, it does not always give the most pleasing board.

The wargame package does not, in and of it self, provide fancy "modern" graphics (though it can be done). Rather, off the shelf, it mimics classic wargames of yore (Afrika Korps, D-Day, Russian Campaign, and so on). That is, it uses rather simplified graphics.

For our game, we will have three kinds of terrain: Clear, woods, and mountains. The map will have a lot of coastline, but the scale of the game is such that naval units are not really called for. Indeed, the game focuses on land combat with abstracted aerial support.

The first thing we do, is to decide a few things about the map. We want to have the hexes automatically label with alphabetic columns and numerical rows. We want to start our rows and columns at 1 (the default is starting at 0). By default, we want the hexes to be white (for clear terrain). To set this up, we define some more keys.

```
\tikzset{%
  hex/short bottom columns=even,
  hex/short top columns=odd,
  hex/label is name,
  hex/first row and column are=1,
  every hex/.style={
    /hex/label={auto=alpha column},
    fill=white},
}%
```

Note how we used the *full* path for the `/hex/label` key. This will be important when we generate a VASSAL module later on.

The key `hex/label is name` makes it so that every hex on the board will be a named `node`. This means we can refer to a hex in Ti*k*Z/PGF simply by its label. In fact, we can use various anchors of the hex if we want to — it is a `node` like any other node in Ti*k*Z/PGF.

The keys `hex/short bottom column=even` and `hex/short to` is mainly used by the `\boardframe` macro to generate a proper frame.

Having defined the settings, we can move on to the board itself. We will do this in parts. This makes it easier to get an overview of, and helps us in this tutorial to explain what is going on.

The first thing we do, is define a macro that defines the coast line of our map. We use the hexagonal coordinate system provided by the wargame package, but we could also have used general Cartesian coordinates.

> Defining the coastline of a map can be quite tedious. You may want to use some vector graphics editor, say Inkscape, to do this. In the wargame package you will find a Python script to convert an SVG to Ti*k*Z/PGF paths.

Figure 13: The coast line

Alright, let's make our macro to define the coastline.

```
\newcommand\coastline{%
  (hex cs:c=1,r=1,v=SE)
  --(hex cs:c=1,r=1,e=SE)
  --(hex cs:c=1,r=1,e=NE,o=.5)
  --(hex cs:c=1,r=1,v=NW,o=.7)
  --(hex cs:c=1,r=2,v=W,o=.6)
  --(hex cs:c=1,r=2,v=NW,o=.7)
  --(hex cs:c=1,r=3,v=SE,o=.9)
  --(hex cs:c=1,r=3,e=SE,o=.9)
  --(hex cs:c=2,r=4,e=S,o=.7)
  --(hex cs:c=2,r=3,v=NE,o=.4)
  --(hex cs:c=2,r=3,e=S,o=.1)
  --(hex cs:c=2,r=3,v=E,o=.8)
  --(hex cs:c=2,r=3,e=SE,o=.8)
  --(hex cs:c=2,r=3,v=SE,o=.6)
  --(hex cs:c=2,r=2,v=NW,o=.2)
  --(hex cs:c=2,r=2,e=SE)
  --(hex cs:c=3,r=1,e=NE)
  --(hex cs:c=4,r=2,e=N,o=.3)
  --(hex cs:c=5,r=2,e=S,o=.6)
  --(hex cs:c=5,r=2,v=SE,o=.9)
  --(hex cs:c=7,r=3,v=E)
  --(hex cs:c=7,r=1,v=E)
  --(hex cs:c=6,r=1,v=E,o=2.5)
  --cycle
  (hex cs:c=3,r=2,v=W,o=.5)
  --(hex cs:c=4,r=3,e=NW,o=.75)
  --(hex cs:c=4,r=3,e=SW,o=.3)
  --(hex cs:c=4,r=2,e=NW,o=.7)
  --(hex cs:c=3,r=2,v=SW,o=.4)
  --cycle
  (hex cs:c=2,r=4,e=N,o=.8)
  --(hex cs:c=2,r=4,v=E,o=.7)
  --(hex cs:c=3,r=3,v=E,o=.8)
  --(hex cs:c=4,r=3,v=NW,o=.6)
  --(hex cs:c=4,r=3,e=NW,o=.1)
  --(hex cs:c=4,r=3,v=SE,o=.4)
  --(hex cs:c=5,r=3,v=SW,o=.6)
  --(hex cs:c=5,r=3,v=NE,o=.2)
  --(hex cs:c=5,r=3,e=NE)
  --(hex cs:c=6,r=4,v=SW,o=.3)
  --(hex cs:c=5,r=4,v=E,o=1.7)
  --(hex cs:c=6,r=5,e=N)
  --(hex cs:c=6,r=6,v=E)
  --(hex cs:c=7,r=6,v=NW,o=.4)
  --(hex cs:c=6,r=7,e=S,o=.2)
  --(hex cs:c=6,r=7,e=NW)
  --(hex cs:c=6,r=8,e=NW,o=.7)
  --(hex cs:c=7,r=9,e=NE)
  --(hex cs:c=6,r=10,v=E,o=2.5)
  --(hex cs:c=7,r=10,v=E)
  --(hex cs:c=1,r=10,v=W)
  --(hex cs:c=2,r=4,v=W,o=2.5)
  --(hex cs:c=2,r=5,v=SW)
  --cycle
  ;
}
```

The coast line is shown in Figure 13[10]

As you can see, it is pretty simple. Not too many details

[10] Any resemblance to any actual location is, erhm, accidental?

about the nooks and crannies. Now we want to define some colours that we will use on the map. As mentioned above, we will have some sea, and we will have some surrounding neutral countries. Let us define some colours for these things.

```
\definecolor{sea}{HTML}{18b5db}
\colorlet{neutral}{gray!25!white}
\colorlet{coast}{black}
```

OK, with that out of the way, we start on the hexes in earnest. We define the macro `\hexes` to produce our hexes within our map.

```
\newcommand\hexes[1][]{
```

We have defined this macro to take a single optional argument. We wont use that here, but it might come in handy at some later point.

The first thing we do, is to fill the entire map with `sea`.

```
  \fill[sea](-1.000,-0.866) rectangle(10.,15.580);
```

Then we use the macro `\coastline` to define a path that we may use over and over again.

```
  \path[save path=\coast]\coastline;
```

Next up, we want to draw hexes that are on the coast line but also has land parts in it. This is so that the hexes are complete and get a proper label on it. We define the fill to be the sea colour, and the lines labels to be white

```
  \begin{scope}[
    every hex/.append style={white,fill=sea},
    hex/label/.append style={color=white}]
    \hex(c=1,r=1);    \hex(c=1,r=2);
    \hex(c=1,r=3);
    \hex(c=2,r=2);    \hex(c=2,r=3);
    \hex(c=2,r=4);
    \hex(c=3,r=2);    \hex(c=3,r=3);
    \hex(c=4,r=2);    \hex(c=4,r=3);
    \hex(c=5,r=3);    \hex(c=5,r=7);
    \hex(c=6,r=4);    \hex(c=6,r=5);
    \hex(c=6,r=6);    \hex(c=6,r=7);
    \hex(c=6,r=8);    \hex(c=6,r=9);
    \hex(c=7,r=6);    \hex(c=7,r=9);
  \end{scope}
```

Note that we use the `.append style` handler so that we get the setting from above here too (labels, among other things). These amended styles only have effect inside the `scope`. Once we leave the scope, then the old settings are restored.

We draw a compass needle.

```
  \draw[white,-{Stealth[]},
    scale line widths,
    line width=1mm]
    (hex cs:c=7,r=4,v=SE) -- ++(110:2)
    node[midway,
      font=\sffamily\bfseries\LARGE,
      transform shape,
      rotate=-90,sloped]{N};
```

Since the areas where we do not add hexes are considered neutral countries in this game, we fill the whole coast line with the previously defined `neutral` colour.

```
  \settosave{\coast}
  \fill[neutral];
```

Here we have the first use of the saved path `\coast`. We make that the current path by calling `\setosave` — a small utility in the wargame package.

We can now add the actual hexes. Some of the hexes will represent different terrain, and that will be clear from the definitions. We will put our definitions inside a scope, and the first thing we do in that scope is to clip everything to our coast line. This will be the second use of `\coast`.

```
  \begin{scope}
    \settosave{\coast}
    \clip;

    \hex(c=1,r=1);\hex(c=1,r=2);\hex(c=1,r=3);
```

Note that we use the `hex cs` coordinate system. We specify the column (`c`) and the row (`r`) of each hex. We can use longer forms, of the keys if we want.

The southern border of A factions country needs a little special attention. We clip the drawn hex, and draw the border. In fact, we draw the hex twice, first time with a white outline and the neutral fill colour. This is so that the full hex will be drawn, even it is only partially within the map area.

```
    \hex[white,fill=neutral,label=](c=2,r=2);
    \begin{scope}
      \clip(hex cs:c=2,r=2,v=SW)
      --(hex cs:c=2,r=2,v=E)
      --(hex cs:c=2,r=2,v=NE)
      --(hex cs:c=2,r=2,v=NW)
      --(hex cs:c=2,r=2,v=W)
      --cycle;
      \hex(c=2,r=2);
    \end{scope}
    \draw[coast] (hex cs:c=2,r=2,v=SW)
    --(hex cs:c=2,r=2,v=E);
```

Note that our clipping path is defined in the `hex cs` coordinate system, and how we have retrieved vertex coordinates via the key `v`.

Now we come to some other terrain: woods and mountains.

```
    \hex(c=2,r=3);\hex(c=2,r=4);\hex(c=2,r=5);
    \hex[terrain=mountains](c=2,r=6);
    \hex[terrain=mountains](c=2,r=7);
    \hex[terrain=mountains](c=2,r=8);
```

Again, we do not want to draw the hex in the neutral country to the south, so we clip this hex too.

```
    \hex[white,fill=neutral,label=](c=4,r=2);
    \begin{scope}
      \clip(hex cs:c=4,r=2,v=W)
      --(hex cs:c=4,r=2,v=NE)
      --(hex cs:c=4,r=2,v=NW)
      --cycle;
      \hex(c=4,r=2);
    \end{scope}

    \hex(c=3,r=2); \hex(c=3,r=3);
    \hex[terrain=woods](c=3,r=4);
    \hex(c=3,r=5);
    \hex[terrain=woods](c=3,r=6);
    \hex[terrain=woods](c=3,r=7);
    \hex[terrain=mountains](c=3,r=8);
    \hex[terrain=mountains](c=3,r=9);
```

In the next step, we will add a town to the map.

```
    \hex[town={
      place={(hex cs:c=1,r=1,e=SW,o=.5)}}]
      (c=4,r=3);
    \hex[terrain=woods](c=4,r=4);
    \hex(c=4,r=5);
    \hex[terrain=woods](c=4,r=6);
    \hex[terrain=woods](c=4,r=7);
    \hex[terrain=woods](c=4,r=8);
    \hex[terrain=woods](c=4,r=9);
    \hex(c=4,r=10);
```

The `place` key specifies the placement of the town relative to the hex centre.

For the rest of the map, there isn't much new.

```
    \hex(c=5,r=3);
    \hex[terrain=woods](c=5,r=4);
    \hex[](c=5,r=5);
    \hex[terrain=woods](c=5,r=6);
    \hex[terrain=woods](c=5,r=7);
    \hex[terrain=woods](c=5,r=8);
    \hex[terrain=woods](c=5,r=9);

    \hex[terrain=woods](c=6,r=4);
    \hex[terrain=woods](c=6,r=5);
    \hex[town={
      place={(hex cs:c=1,r=1,v=E,o=.3)}}]
      (c=6,r=6);
    \hex(c=6,r=7); \hex(c=6,r=8);
    \hex[terrain=woods](c=6,r=9);
    \hex[terrain=woods](c=6,r=10);

    \hex(c=7,r=4); \hex(c=7,r=5);
    \hex(c=7,r=6); \hex(c=7,r=7);
    \hex(c=7,r=8); \hex(c=7,r=9);
  \end{scope}
}
```

That concludes our hexes. Let us draw it (Figure 14).

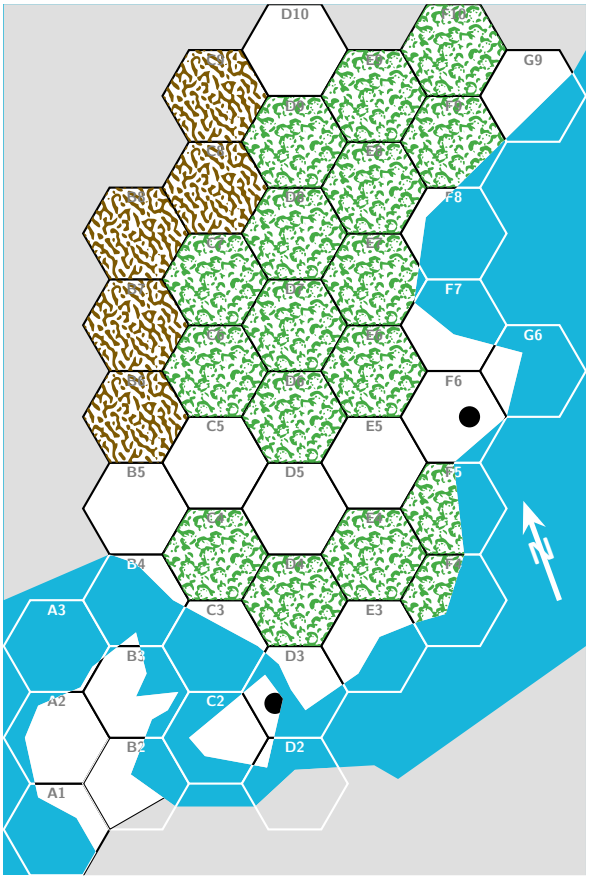


Figure 14: Our hexes

This only defines the map. Now we would like to make the full board. Typically one want to add a turn track and some other stuff. Here, we will keep it fairly simple and just add a turn track.

> This is where we need to start to consider VASSAL. Essentially, VASSAL can split a board into *zones*. These zones can be different game specific places, e.g., a place to keep eliminated units, a place for the turn track, and so on. We will take advantage of this in our board definition by applying some specific styles.

Below we customise the board frame, and some other things on the board.

```
\colorlet{framebg}{gray!50!white}
\colorlet{titlefg}{gray!50!black}
\tikzset{
  hex/board frame/.style={
    scale line widths,
    line width=.5pt,
    draw=black},
  board frame/.style={%
    anchor=south west,
    transform shape,
    scale line widths,
    line width=1pt,
    draw=black,
    fill=framebg,
    minimum width=14cm,%
    minimum height=19cm
  },
  turn/.style={
    scale line widths,
    line width=.5pt,
    fill=white,
```

```
    draw=black,
    text=gray,
    minimum width=1.4cm,
    minimum height=1.4cm,
    text width=1.2cm,
    align=center,
    font=\sffamily\bfseries\huge,
    transform shape,
    anchor=north east
  },
  title/.style={
    text=titlefg,
    font=\sffamily\bfseries\Huge,
    scale=1.5,
    inner sep=0pt,
    outer sep=0pt,
    transform shape,
  },
  sub title/.style={
    text=titlefg!75!white,
    font=\sffamily\bfseries\normalsize,
    scale=1.5,
    transform shape,
    text width=5cm,
    inner sep=0pt,
    outer sep=0pt,
    align=right
  }
}
```

Let us start the board. The wargame package has a tool `\boardrame` which will put a frame around the map as defined by hex coordinates. It also output the map bounding box to the standard log file. So running LaTeX once with this macro in effect will give us some useful information.

We will define the board in two steps. First we define an *environment* which holds the map, turn track, and title of the game, and after that we will add an environment that wraps it in a Ti*k*Z/PGF picture.

We define the board as an environment, so that we may put things on the board in our explanations of the rules.

```
\newenvironment{innerboard}[1][]{%
  \node[board frame] (frame) {};
```

This adds in our frame of the whole map as defined by the style above.

Also note that we made our outer frame a `node`. This is so that we can refer to its anchors without too much knowledge of its size.

We have made our frame a little bigger than the hex, so that we may add a turn track. We will at it at the top, and we will add 10 turns. Here, we make use of Ti*k*Z/PGF loops, and they do the trick for us.

We will make use the style `zone scope` with the name `Turns`. This will be made into a VASSAL zone that has a coordinate system that identifies it (at least by convention) as turn track. We will in fact deploy a little trick here to make the module even more complete. This is because we add the style `zone point` to the turn number nodes below, and the first one will have the name of the game turn marker. This means that the game turn marker will *automatically* be placed there. If the below is a little complicated for you, take a look at the alternative below the code

```
  \begin{scope}[
    shift={($(frame.south east)+(-0.5,.5)$)}]
    \begin{scope}[zone scope=Turns]
      \foreach \i in {1,...,10}{%
        \pgfmathparse{\i*1.65-.225-.7}
        \edef\y{\pgfmathresult}
        \ifnum1=\i
          \def\tname{game turn}
        \else
          \def\tname{T\i}
        \fi
        \node[turn,zone point={\tname}{-.7}{\y}]
        at (0,\y+.7) {\i};
      }
    \end{scope}
  \end{scope}
```

A simpler alternative would be

```
\begin{scope}[
  shift={($(frame.south east)+(-0.5,.5)$)}]
  \begin{scope}[zone scope=Turns]
    \foreach \i in {1,...,10}{%
      \node[turn] at (0,\i*1.65-.225) {\i};
    }
  \end{scope}
\end{scope}
```

We also add a title using the styles we defined above. Note, consistent use of Ti*k*Z/PGF styles makes it very easy to change things consistently throughout the game.

```
\node[below right=5mm and 5mm of
frame.north west,title]{A Game};
\node[below left=5mm and 5mm of
frame.north east,sub title]{A tutorial in the\\
   {\color{white}wargame} package};
```

With this, we are left with adding in the map.

```
\scope[shift={(.5,.5)}]
  \scope[shift={(1,0.86603)},
    zone scope=Hex]
    \hexes
    \boardframe(c=1,r=1)(c=7,r=10)
    % Reported by the above
    % Board Frame:
    %   BBox:  (-1.0,-0.86603)x(10.0,15.5885)
    %   WxH:   (11.0x16.45453)
    %   NCxNR  (1,1)x(7,10)
```

Above, we see the output from `\boardframe`. This in turn informs us how to set the offset of the inner most scope. We also see that the map is 11×16.45453 cm big, which means our board will easily fit on standard paper (A4 or, less standard, Letter).

The offset of $(-1, 0.86603)$ is not surprising when you know that the radius r of the circumscribed circle of the hexes is 1 cm, and that the first hex centre is placed in $(0, 0)$. The horizontal offset of -1 is just that radius r, and the vertical offset is $r \sin 60° = \sqrt{3}/2 = 0.86603$.

The outer most scope translates the hex 5mm in from both edges so that we may draw a nice frame around the map.

The style `zone scope=Hex` will make this area a zone in VASSAL and it will have a hex grid (because its argument contains the sub string `hex`) attached to it.

This end the start of the environment. Next, we define the end of it. This simply closes the scopes and the Ti*k*Z/PGF picture.

```
}{
    \endscope%
  \endscope%
}
```

This ends the environment that defines the board proper. Now we wrap it in another environment that will put the map inside a Ti*k*Z/PGF picture.

```
\newenvironment{board}[1][]{%
  \tikzpicture[#1,zoned]
  \innerboard}{%
  \endinnerboard%
  \endtikzpicture}
```

The style `zoned` on the Ti*k*Z/PGF picture ensures that we will get a zoned map when generating the VASSAL module. Note that we used the TEX-like form of the environment `\tikzpicture`, rather than `\begin{tikzpicture}`, so that the various `\end`s does not mess up things.

We can draw the final board (Figure 15).

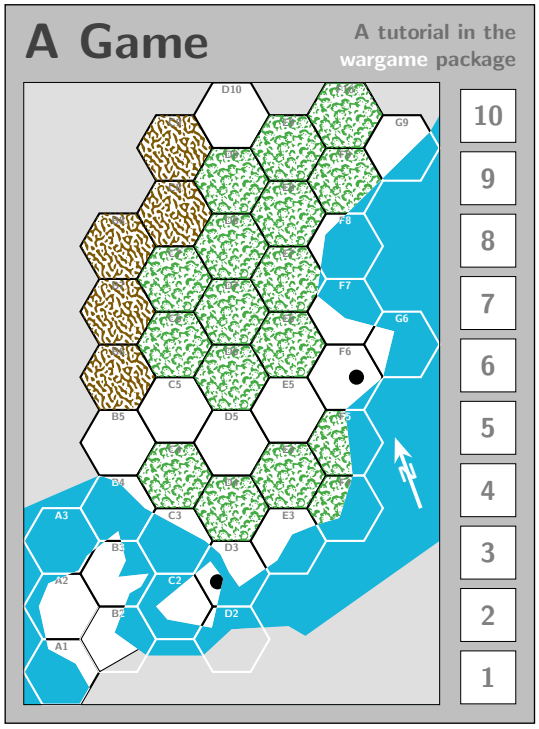


Figure 15: The final board

5.1 Adding units to the board

We have defined our board as a environment so we can add units to it without much hassle. Let us add the starting units at the starting positions. Since we will be *stacking* chits, we will use the `\chitstack` macro for this. Note that this macro expects code to produce the chits (rather than the unit styles). Therefore we must add in some `\noexpand` in front of the `\chit` command.

The board with the "At-start" units in place are shown in Figure 16. The first part of this was made with

```
\begin{board}
  \stackchits(A2)(.2,.2){%
    \noexpand\chit[a hq],
    \noexpand\chit[a 1 hqbg],
```

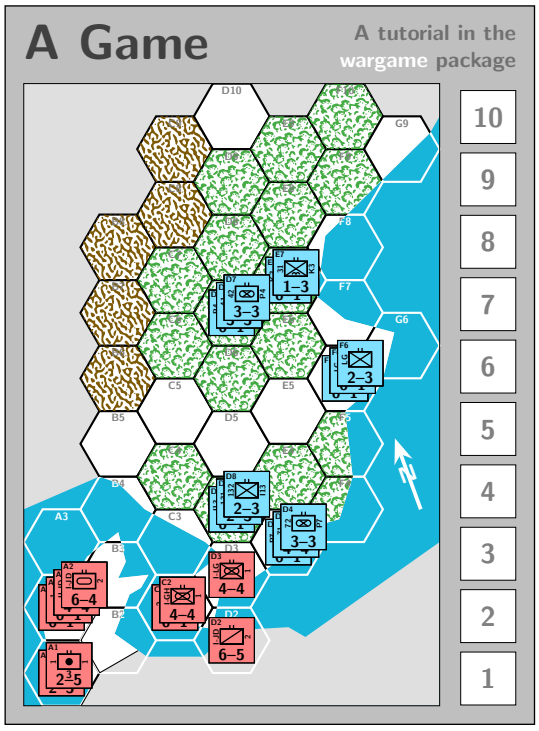


Figure 16: The board with units at their starting positions

```
    \noexpand\chit[a 1 2jd ibn],
    \noexpand\chit[a 2 1jd abn]}
  \chit[a 1 1lg ibn](D3);
```

The first argument to `\stackchits` is where to put the stack. The second is the offset applied to each stack unit in succession (obviously the first offset is $(0,0)$). The third argument is the list of chits to add to the stack.

Note that we use the hex labels, rather than the hex coordinates, to place the units and stacks of units, thanks to `hex/label is name`.

We have made the counters and board for the game. What remains is the various tables (and or of course the rules).

5.2 Clipped boards

We are not quite done with the board just yet. We want to make *another* environment which will clip to a specified area of the map. We will reuse our `innerboard` environment so that everything is consistent. Let us go ahead and define the environment, and then see later how it is used. We define our environment as two macros `\clipped` and `\endclipped` so that we can use parenthesis for our arguments.

```
\tikzset{
  clip board/.style={gray,line width=.5pt}}
\newcommand\clipped[1][]{\doclipped[#1]}
\def\doclipped[#1](#2)(#3){
  \def\tmp@clip@a{#2}
  \def\tmp@clip@b{#3}
  \tikzpicture[#1]
    \scope
      \clip (#2) rectangle (#3);
      \scope[shift={(-.5-1,-.5-0.86603)}]
        \innerboard}
\def\endclipped{%
        \endinnerboard
      \endscope
    \endscope
    \draw[clip board](\tmp@clip@a)
    rectangle (\tmp@clip@b);
  \endtikzpicture}
```

Our environment `clipped` takes 3 arguments. One is optional and is keys to pass to the Ti*k*Z/PGF picture environment. The two others are the coordinates (relative to the hexes) of our clipping rectangle. Let us illustrate this with an example — see Figure 17.

The figure was produced by the code

```
\begin{clipped}(hex cs:c=2,r=2)(hex cs:c=5,r=5)
\end{clipped}
```

Note that we *must* use the hex coordinate system. When we make the clipping path, the nodes of the hex map has not yet been defined. We can use this to illustrate things on the map. For example, Figure 18 show that B P7/72 CABN attacks A 2/1JD ABN with the code

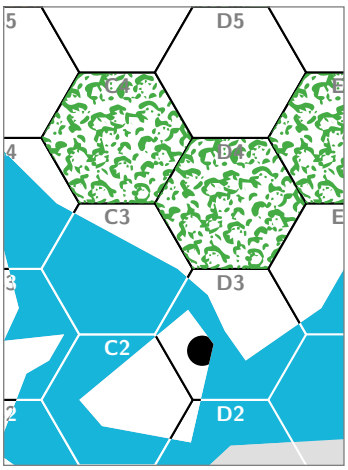


Figure 17: Example of using `clipped`.

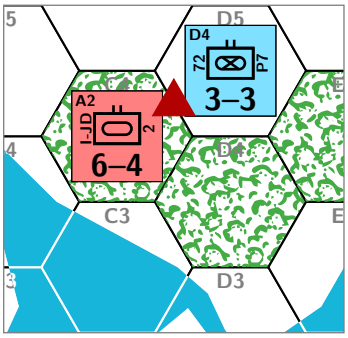


Figure 18: Example of using `clipped`.

```
\begin{clipped}(hex cs:c=2,r=3)(hex cs:c=5,r=5)
  \chit[b p7 72 cabn](D5);
  \chit[a 2 1jd abn](C4);
  \draw[hex/attack] (D5)--(C4);
\end{clipped}
```

Note that *inside* the environment we *can* use hex labels as coordinates.

6 The charts

We will not go into details about how to make tables in LaTeX. There are plenty of excellent resources out there for that purpose. Here we will make three tables: The combat resolution table (CRT), the terrain effects chart (TEC), and the Order of Battle (OOB) chart[11] — all of which in some respect uses elements of the wargame package.

[11] In some games this is called the Order of Appearance (OOA) chart.

Die roll	Odds					
	1:3	1:2	1:1	2:1	3:1	4:1
1	AE	AR	AR	EX	DR	DR
2	AE	AR	EX	EX	DR	DR
3	AR	EX	EX	DR	DR	DE
4	AR	EX	EX	DR	DR	DE
5	EX	EX	DR	DR	DE	DE
5	EX	DR	DR	DE	DE	DE

Table 1: The combat resolution table

For all the tables, we would like to use the colortbl package to colour the rows. Here, we make some short hands for doing just that.

```
\colorlet{headbg}{gray!50!white}
\colorlet{altbg}{gray!15!white}
\newcommand\headrow{\rowcolor{headbg}}
\newcommand\defrow{\rowcolor{white}}
\newcommand\altrow{\rowcolor{altbg}}
```

6.1 The CRT

The combat resolution table is in some sense a straight forward LaTeX table. Here's the definition

```
\newcommand\crt{%
  \begin{tabular}{|c|cccccc|}
    \hline
    \headrow
    \textbf{Die}
    & \multicolumn{6}{c|}{\textbf{Odds}}
    \\
    \headrow
    \textbf{roll}
    & \textbf{1:3}
    & \textbf{1:2}
    & \textbf{1:1}
    & \textbf{2:1}
    & \textbf{3:1}
    & \textbf{4:1}
    \\
    \hline
    \defrow
    1 & AE & AR & AR & EX & DR & DR \\
    \altrow
    2 & AE & AR & EX & EX & DR & DR \\
    \defrow
    3 & AR & EX & EX & DR & DR & DE \\
    \altrow
    4 & AR & EX & EX & DR & DR & DE \\
    \defrow
    5 & EX & EX & DR & DR & DE & DE \\
    \altrow
    5 & EX & DR & DR & DE & DE & DE \\
    \hline
  \end{tabular}}
```

And what it looks like is shown in Table 1

6.2 The TEC

In this table we will use small version of hexes to make a nice table. The definition is as follows.

```
\newcommand\tec{%
  \begin{tabular}{|cl|cc|}
    \hline
    \headrow
    \multicolumn{2}{|c|}{\textbf{Terrain}}
    & MF
    & CF\textsuperscript{\dag}
    \\
    \hline
    \defrow
    \tikz[scale=.5]{\hex[label=]}
    & Clear
    & 1
    & ---
    \\
    \altrow
    \tikz[scale=.5]{\hex[label=,terrain=woods]}
    & Woods
    & 2
    & \texttimes 2\textsuperscript{*}
    \\
    \defrow
    \tikz[scale=.5]{\hex[label=,terrain=mountains]}
    & Mountains
    & Stop
    & \texttimes 2\textsuperscript{*}
    \\
    \altrow
    \tikz[scale=.5]{
    \hex[label=,fill=sea](c=1,r=1);
    \begin{scope}
      \clip
      (hex cs:c=1,r=1,v=NW)
      --(hex cs:c=1,r=1,v=W)
      --(hex cs:c=1,r=1,v=SW)
      --(hex cs:c=1,r=1,e=S)
      --(hex cs:c=1,r=1,v=W,o=.5)
      --cycle
      (hex cs:c=1,r=1,e=N)
      --(hex cs:c=1,r=1,v=NE)
      --(hex cs:c=1,r=1,v=E)
      --(hex cs:c=1,r=1,e=SE)
      --(hex cs:c=1,r=1,v=E,o=.4)
      --cycle
      ;
      \hex[label=](c=1,r=1);
    \end{scope}
    }
    & Coastal
    & 2
    & \texttimes2\phantom{\textsuperscript{*}}
    \\
    \hline
    \multicolumn{4}{l}{\textsuperscript{\dag}
      \emph{Defender} modifier}\\
    \multicolumn{4}{l}{\textsuperscript{*}
      \infantrymark{}\,\pgmark{} \emph{only},
      unless \artillerymark}
    \\
  \end{tabular}
}
```

Terrain	MF	CF†
Clear	1	—
Woods	2	×2*
Mountains	Stop	×2*
Coastal	2	×2

† *Defender* modifier
* ⊠⊠ *only*, unless ⊡

Table 2: The terrain effects table

The result is in Table 2. Note the use of `\infantrymark` to put in ⊠ in the table. The wargame package defines a number of macros like this.

6.3 The OOB

We like to make an OOB chart that we can place our unit counter on so we can keep track of which units are available when. In our game, all A faction units start already on the board. The B faction, however, has most of its reinforcement troops way up north, and will therefore bring them on the map piecemeal.

We will use the wargame macro `\oob` to make our OOBs. This macro uses the `pic chits/oob turn` to make the turns. The default setting is not what we want, so we redefine that picture. In particular, we want turn "0" to say "At start".

We also want to reuse the style `turn` we used to make the turn track on the board, but modify it a bit to fit our purposes here. We therefore define the style `oob turn` which derives from `turn`, but scale it down 70%, change the anchor, and add some extra space outside of it (`outer sep`).

```
\tikzset{
  oob turn/.style={
    turn,
    transform shape,
    anchor=east,
    outer sep=2mm,
    scale=.7},
}
```

Our modified `chit/oob turn` picture will check if we get an empty turn number, or the special "0" turn, and then put in a node with the appropriate contents. Note that we define two pictures, one which actually puts in the turn, and one that puts in nothing, so when we make the combined OOB we only output the turns once. We then default `chit/oob turn` to the first picture

```
\tikzset{
```

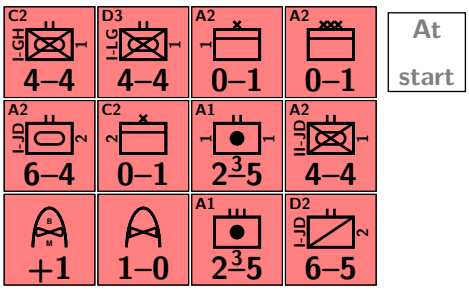


Figure 19: Faction A OOB

```
  oob turn real/.pic={%
    \def\tmp{{\Large At start}}
    \ifx|#1|\else%
      \ifnum0=#1\else%
        \def\tmp{#1}%
      \fi%
    \fi%
    \node[oob turn,transform shape]{\tmp};
  },
  oob turn empty/.pic={},
}
\newcommand\oobrealturn{%
  \tikzset{
    pics/chit/oob turn/.style=%
    {oob turn real=##1}}}
\newcommand\oobemptyturn{%
  \tikzset{
    pics/chit/oob turn/.style=%
    {oob turn empty=##1}}}
\oobrealturn
```

Let us see how this looks for both factions. Factions A and B are shown in Figure 19 and 20, respectively.

These were done with

```
\begin{tikzpicture}[oob turn/.append style={anchor=west}]
  \oob*{\alla}{4}{1.24}{0}
\end{tikzpicture}
\begin{tikzpicture}
  \oob{\allb}{4}{1.24}{0}
\end{tikzpicture}
```

Note that the stared version `\oob*` puts out the counters right aligned. This is also why we append the style `oob turn` to be anchored to the west.

The first argument to `\oob` is the list of lists of counters. This is why we defined `\alla` and `\allb` in the way we did. The second argument is the distance between each counter. Since the counters are 1.2 cm wide, we gave them a little extra space between them by setting the second argument to 1.24 (cm). The third argument is the spacing between turn rows. Above we have set that to 0 (cm), but that we will change in the real OOB chart.

And that's what we will do next. This will be rather wide, so we when we show it in a minute, we will do so in a wide table (leaving the two-column layout for a

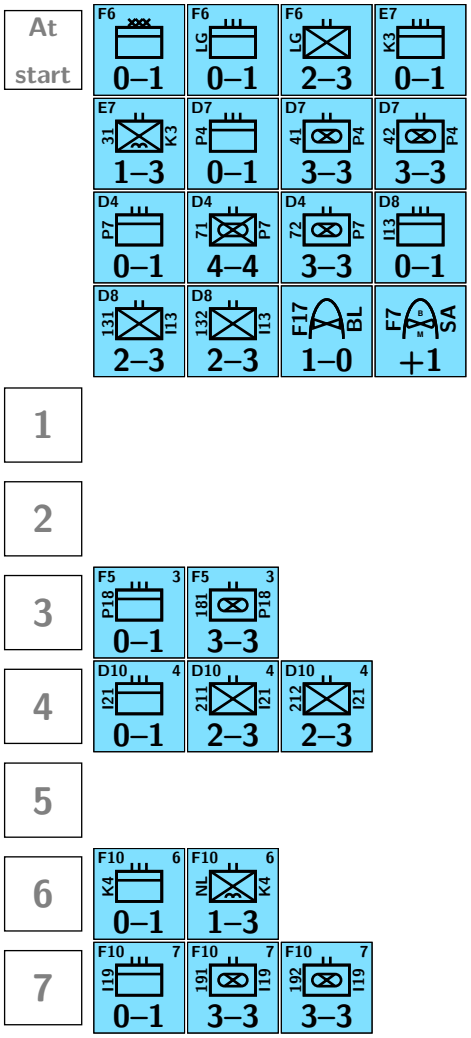


Figure 20: Faction B OOB

minute).

```
\newcommand\inneroob[1][]{%
  \node[board frame,anchor=north] (oob frame)
  at (0,2.5) {};%
  % Game title
  \node[%
    below right=5mm and 5mm of oob frame.north west,
    title,scale=.8]{A Game};
  % Chart title
  \node[%
    below left=5mm and 5mm of oob frame.north east,
    sub title,scale=1.2,transform shape]{%
    Order of Battle};
  % Turn title
  \node[font=\sffamily\bfseries\Large,
    anchor=south,
    white,transform shape] at (0,1) {Turn};
  %
  \begin{scope}[%
      transform shape,
      oob turn/.append style={anchor=west},%
      shift={(-1.4,0)},
      zone scope=A OOB]%
    \oobemptyturn%
    \oob*{\alla}{4}{1.4}{.2}%
  \end{scope}%
  %
  \begin{scope}[%
      transform shape,
      shift={(1.4,0)},
      zone scope=B OOB]%
    \oobrealturn%
    \oob{\allb}{4}{1.4}{.2}%
  \end{scope}%
  %
}
\newcommand\fulloob[1][]{%
  \begin{tikzpicture}[#1,zoned]%
    \inneroob
  \end{tikzpicture}}
```

A couple of things to note.

We put everything inside a Ti*k*Z/PGF picture with the option `zoned` so that we will get a zoned map in VASSAL. We also add the zones `A OOB` and `B OOB` via the key `zone scope`. We will need to adjust the grids of the zones later on. Another side effect of using `\oob` is that each turn will be marked as a "region" of a "region grid" in in VASSAL, so that the each counter will be automatically placed in its place on the OOB in the VASSAL module.

Secondly, we reuse our `board frame`, `title`, and `sub title` styles to make the OOB come out in similar manner as the board. We do adjust the font sizes a bit though. We have also made the OOB the same size as the board so that they stack nicely.

Finally, when we make the faction A oob, we turn off generation of the turn numbers. This is so we do note produce those twice.

The final OOB is shown in Figure 21. This was produced

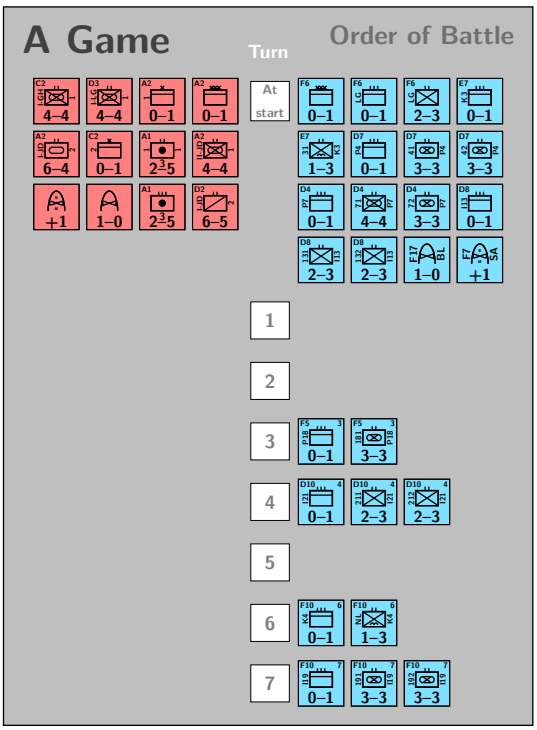


Figure 21: The OOB

with

```
\fulloob
```

7 Ending the package

At the end of the package we should end with the marker `\endinput`.

```
\endinput
```

This is the end of the package. We will do a little more in the main document.

8 Rules

We will not write any rules for this demonstration game. The focus of this tutorial is how to use the wargame package, not how to design a game.

The wargame package does however have a number of tools that helps the developer write the rules. We've already seen some of them in action. For example, we saw how to add stacks of units to a board in Section 5.1 or how to make board clippings to show a zoom of the map (Section 5.2).

Figure 18 showed how we can illustrate an attack, for example. The package also provides means to indicate eliminated units, movement, retreats, advances, and much more. The manual gives a lot more information about this.

9 VASSAL module

As already mentioned several times in this tutorial, we can use the definitions of maps, counters, and charts to generate a VASSAL module.

> VASSAL[12] is a cross-platform[13] tool for playing wargames against other humans or solitaire. One can do Play-By-E-Mail (PBEM) or online.

The trick is to prepare a separate LaTeX document, say `export.tex` as in this tutorial, which exports the images as single pages. Then, the Python script `wgexport.py` distributed with the wargame package picks up this output (`export.pdf` and `export.json`) and generates a draft VASSAL module.

In addition, you can supply a Python script, say `patch.py`, which will be run on the draft module so that you fix things up in the VASSAL module. In this way, your VASSAL module will likely need very little hand editing (if at all), and will be robust against changes in the LaTeX code[14]. This of course requires a bit of familiarity with Python, which is time well spent.

Let us get into it.

9.1 The `export.tex` file

.

This file will output all the images to a PDF, ⟨*export*⟩`.pdf` along with meta information to a JSON[15] file ⟨*export*⟩`.json`. This is relatively simple to do.

We need to use the document class `wgexport`, so lets go a head and load that.

```
\documentclass[11pt]{wgexport}
```

Make sure you give the right font size as argument.

This is where it pays off that we put all our definitions of counters, board, charts, etc. into a separate package. To import these, we simply use that package here[16]. So we load the package and start our document.

```
\usepackage{game}
\begin{document}
```

[12] https://vassalengine.org

[13] It is written in Java, meaning it runs almost everywhere, except on iOS and Android

[14] Assuming you keep your Python `patch.py` script up to date.

[15] JavaScript Object Notation

[16] If we hadn't made that package, we would have to repeat the definitions in some way. Very error prone

Now the body of the document will simply consist of a list of images, one for each side of each counter, the board, charts, and so on. We need to put everything into the environment `imagelist` which will keep track of our meta data.

```
\begin{imagelist}
```

So far so good. The first thing we will output are the counters. For that we will use the macro `\doublechitimages` (if we had single sided counters, we would use `\chitimages`). This macro takes two arguments: The list of counter definitions and the faction of the chits. Let's go ahead and make those for factions A and B, and the special faction "Markers" which will only be the game turn counter.

```
\doublechitimages[A][chit drop shadows]{\alla}
\doublechitimages[B][chit drop shadows]{\allb}
\doublechitimages[Markers][chit drop shadows]{{game turn chit}}
```

Again, the macros we made for the counters comes in handy. Note that we have used it twice so far: For the OOBs and here for the VASSAL module. We will, in fact, use them one more time to create the counter sheet.

To automatically make "battle markers" — markers that identify individual battles by placing a numbered marker on top of the combatants, we can use the macro `battlemarkers`. It takes one argument, which is the number of unique markers to make and add. The markers are round yellow circles with a number in them[17].

```
\tikzset{}
\battlemarkers[marker drop shadows]{12}
\tikzset{every battle marker/.style={}}
```

Furthermore, we can add "odds" and battle result markers. Odds markers can be placed via the battle markers context menu, and then later be replaced by result markers via the context menu of the odds markers. This is done via the macros `\oddsmarkers` and `\resultmarkers`. Both macros accept a list of options (odds and results, respectively) with a possible background colour to use in the markers.

```
\oddsmarkers[marker drop shadows]{%
  1:3/red!25!white,%
  1:2/red!15!white,%
  1:1/orange!15!white,%
  2:1/white,%
  3:1/green!15!white,%
  4:1/green!25!white}
\resultmarkers[marker drop shadows]{
  AE/red!50!white,
  AR/red!25!white,
  EX/white,
  DR/green!10!white,
  DE/green!25!white}
```

[17] This can of course be customised as everything else can.

By default, odds markers must be placed "by-hand", but via the VASSAL preferences it is possible to calculate the odds and show them when the battle is declared. By default, this calculation will only be concerned with the combat factors of the involved units, and features such as terrain is ignored. However, with a bit of Python skill, one can flesh out the implementation of the game rules in the optional patch (Python) script. We will not do that here, as a it is a little beyond the scope of this tutorial.

Right, so that made our counter images and the associated meta data. Next thing is to add our board. For that, we use the environment `boardimage`. Inside that environment we must draw the board. The environment takes one optional argument which classifies the board. Meaningful classifications are `board` (default) and `oob` for OOBs. The first mandatory argument is the name of the board (more or less free form text, except any of "`./\'"`" should not be used. The second mandatory argument is a sub-category, and is mainly reserved for future use.

```
\begin{boardimage}{Board}{}
  \begin{board}
  \end{board}
\end{boardimage}
```

We will make another board image for our OOB, since that component should hold counters in VASSAL. We pass the optional argument `oob` in this case.

```
\begin{boardimage}[oob]{OOB}{}%
  \fulloob
\end{boardimage}
```

We have two charts that we would also like to be put in: The CRT and the TEC. These *must* be put into Ti*k*Z/PGF pictures, and we precede those with the macro `\info`. This macro takes three arguments: The name of the next image, the category, and sub-category. Again, the name is more or less free form, and the category in some sense dictates how it will present in the VASSAL module. For charts we should use the category `chart`. The sub-category isn't really used at this point.

```
\info{CRT}{chart}{}
\begin{tikzpicture}
  \node{\crt};
\end{tikzpicture}
%
\info{TEC}{chart}{}
\begin{tikzpicture}
   \node{\tec};
\end{tikzpicture}
```

We also want background images for our pools of eliminated units in VASSAL. This will be enlarged version of main headquarter unit of each faction, overlayed with a white semi-transparent background.

```
\info{A}{pool}{}
\begin{tikzpicture}[scale=7]
  \chit[a hq];
  \fill[white,opacity=.5](-.6,-.6)rectangle++(1.2,1.2);
\end{tikzpicture}
\info{B}{pool}{}
\begin{tikzpicture}[scale=7]
  \chit[b hq];
  \fill[white,opacity=.5](-.6,-.6)rectangle++(1.2,1.2);
\end{tikzpicture}
```

We can add splash page image. We have not defined such an image in the `game` package, but we could have, so we will make it here. Again, it should be a Ti*k*Z/PGF picture. For this, we need to use the category `front`.

```
\info{Splash}{front}{}
\begin{tikzpicture}
  \node[board frame,title,
  minimum width=5cm,minimum height=5cm]
  {A Game};
\end{tikzpicture}
```

We could continue to add more if we wanted to. Basically anything can be added. For example, one might want custom button icons or the like. We will add a few such icons here, mainly to show how the `patch.py` script works.

The wgexport.cls class provides a number of icons we may use for this. These are

We will add these as pictures to our export PDF. First, the image for the pool of units.

```
\info{pool-icon}{icon}{}
\begin{tikzpicture}[transform shape,scale=.4]
  \pic{pool icon};
\end{tikzpicture}
```

The category `icon` has no special meaning.

We also want to add a custom icon for the OOB button. VASSAL has a button icon for the piece inventory which is really quite appropriate, but alas it is already in use, so we will make our own. Here, we use the Ti*k*Z/PGF picture `oob icon` provided by wgexport. This picture needs two arguments: the left hand and right hand sides fill colours. We will use our background colours.

```
\info{oob-icon}{icon}{}
\begin{tikzpicture}[transform shape,scale=.4]
  \pic{oob icon={a-bg}{b-bg}};
\end{tikzpicture}
```

Normally the `wgexport.py` script uses the *undo* image for the flip button. However, that may be a bit confusing, so we will use a custom image for that. We will use the pictures provided by wgexport.

```
\info{flip-icon}{icon}{}
\begin{tikzpicture}[transform shape,scale=.4]
  \pic{flip icon};
\end{tikzpicture}
%
\info{eliminate-icon}{icon}{}
\begin{tikzpicture}[transform shape,scale=.4]
  \pic{eliminate icon};
\end{tikzpicture}
%
\info{restore-icon}{icon}{}
\begin{tikzpicture}[transform shape,scale=.4]
  \pic{restore icon};
\end{tikzpicture}

\end{imagelist}
\end{document}
```

9.2 Make the draft module

We run LaTeX on the above `export.tex` file (the source of this section) to generate `export.pdf` and `export.json`. We then process these with the `wgexport.py` Python script to get the draft VASSAL module `Draft.vmod`.

```
`kpsewhich wgexport.py` export.pdf export.json
```

The utility `kpsewhich`[18] TeXLive and similar TeX distributions. will report the location of a file in the TeX installation tree(s). Note that the ` above are what is typically called 'back-ticks' (i.e., what you typically put as the leading character of scare-crows in LaTeX). This is of course Un*x-like syntax. For other OSs, consult your TeX distribution's documentation for how to find files in the TeX installation.

You can open the module in the VASSAL editor and see what was generated. On the board, all areas marked with the `zone scope` (or `zone path`) should be defined as zones. In the OOB, we should have two zones, one for the A faction and one for the B faction. We should also have three groups of counters — one for each faction and one for "Markers". Finally, we should have a window with tabs, one for each defined chart.

We can add the rules to the module. So suppose the rules are in `game.pdf`, then we can do

```
`kpsewhich wgexport.py` export.pdf export.json \
   -r game.pdf -o Game.vmod
```

to add the rules to the "Help" menu. Other information is added to that menu too, such as key-bindings and a short note on how the module was generated. Of course, you should also see the splash image that we defined. The option `-o` ⟨*filename*⟩ writes the module to ⟨*filename*⟩ rather than the default `Draft.vmod`.

We define the version of the module by passing the option `-v` ⟨*version*⟩ to the Python script. By default, the version is set to `draft`, and a note on the draft is added to the module and the grids are drawn. Any other version string will suppress these.

Also, the default module name is `Draft` and a similar description is likewise the default. We can change this by passing the options `-t` ⟨*title*⟩ and `-d` ⟨*description*⟩. Note, you may want to quote the arguments if they contain spaces or the like. Thus, to make a more complete module, we would do something like

```
‘kpsewhich wgexport.py‘ export.pdf export.json \
  -r game.pdf -o Game.vmod \
  -t "LaTeX Wargame tutorial" \
  -d "Example of using wargame to make a module" \
  -v 0.1
```

Note that you typically need to quote (`"`) longer strings that contain spaces and other special characters.

9.3 Going the extra mile

We will take one more step on the VASSAL module, and then we will get back to the Print'n'Play version of the game. We can provide the `wgexport.py` script with an additional Python script that can tweak the VASSAL module any way we like. We can for example move counters into their starting positions, to the OOB, adjust grids, add more materials, and so on. It is only your imagination, and Python programming skills, that sets the limits on what you can do.

Here we will make a simple `patch.py` Python script which does very little. The wargame package, wgexport class, and `wgexport.py` script has already done the heavy work for us. Not least be because we have taken care to add in `zone` styles where needed.

We will put all the counters on the OOB chart, and adjust some grids - most notably the OOB and turn track grids. Other than that, we will not do much.

```
‘kpsewhich wgexport.py‘ export.pdf export.json \
  -r game.pdf -o Game.vmod
  -t "LaTeX Wargame tutorial" \
  -d "Example of using wargame to make a module" \
  -v 0.1 -p patch.py
```

Now for the script:

```
# --- We may need to import the export module ---
#
# from wgexport import *

def patch(build,data,vmod,verbose=False,**kwargs):
    # --- Get the game ---
    game = build.getGame()

    # --- Get the maps ---
    maps  = game.getMaps()

    # --- Get the main board ---
    board = maps['Board']

    # --- Get the mass keys ---
    mkeys = board.getMassKeys()
    mkeys['Eliminate']['icon'] = 'eliminiate-icon.png'
    mkeys['Flip']     ['icon'] = 'flip-icon.png'

    # --- Get the dead-pool map ---
    pool = maps['DeadMap']
    pool['icon'] = 'pool-icon.png'
    pool.getMassKeys()['Restore']['icon'] = 'restore-icon.png

    # --- Get the OOB map ---
    oob = game.getChartWindows()['OOBs']
    oob['icon'] = 'oob-icon.png'


#
# EOF
#
```

In the `patch.py` script we can use all the functionality provided by `wgexport.py`. Elements are XML elements that we apply `xml.dom.minidom` operations on.

Note, if we need to use classes, etc. from `wgexport.py`, then we ought to import that module into our patch script, as shown in the top comment above. In the example patch script we do not need that, so we leave it out.

One can do quite complicated things in VASSAL, which can be set-up in the patch script. By default, the `wgexport` script sets up the pieces and boards, and if one has defined regions with specific piece names, then the script will likewise place the pieces there (as with our `game turn` example above).

One can, for example, setup the module so that the game turn marker automatically flips or progresses when the turn track widget reach specific points. One can in principle also set it up so that when a specific turn or phase is reach, then pieces are moved from the OOB to the board. The more one can do like that, the more the game is automated.

10 The materials

Next, we will make some materials for printing. The next few pages will contain only the materials and no text, so we will summarise what we do here.

The first thing we put out is the map. That is pretty

simple. We do

```
\begin{board}
\end{board}
```

Then comes the OOB. We simply use the `\fulloob` macro.
After that are the two tables put into one page.

```
\crt \tec
```

And the final part are the counter sheets. We will take advantage of the macros `\alla` and `\allb` yet again.

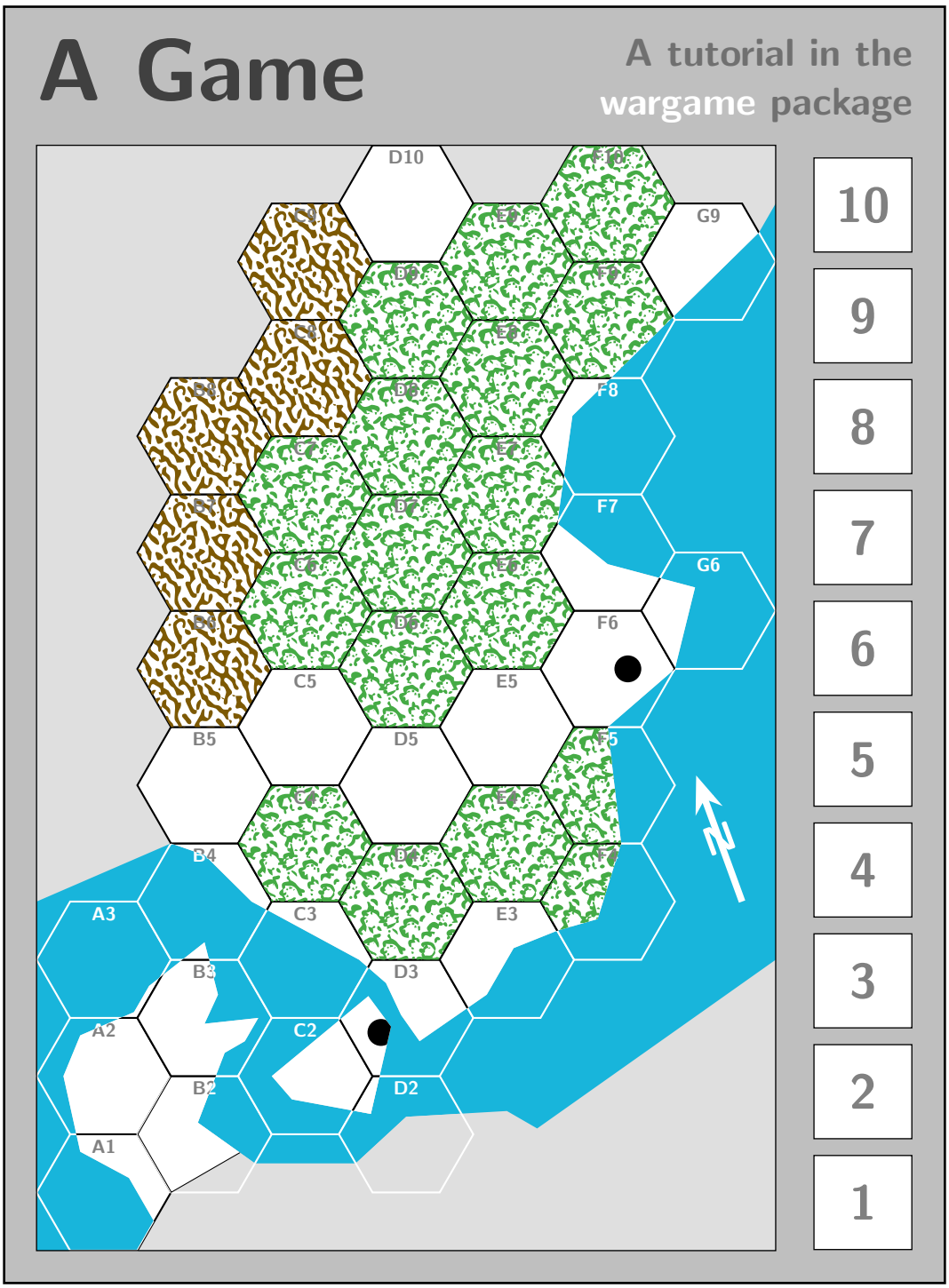
```
\doublechits{\allb}{4}{1.24}
```

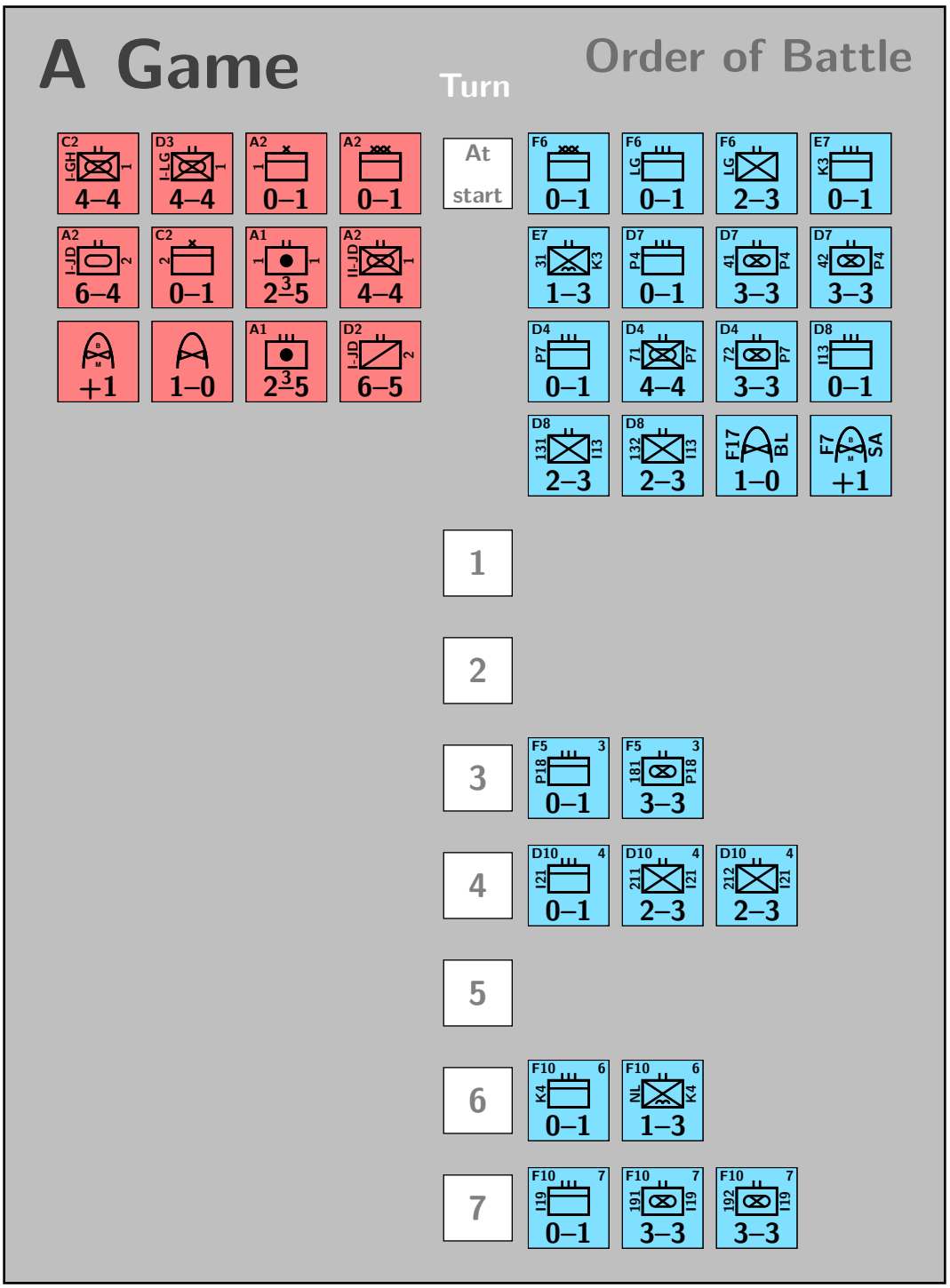
As a final bonus, we also make a board that includes our three charts: The OOB, CRT, and TEC. We will not demonstrate the code here, but take a look in the sources if you are curious.
We put in `\cleardoublepage` between all this so that we can print directly on a duplex printer.

11 Epilogue

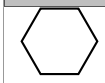
I hope that this short tutorial has helped you. Suggestions, corrections, and so on are very welcomed.

Christian



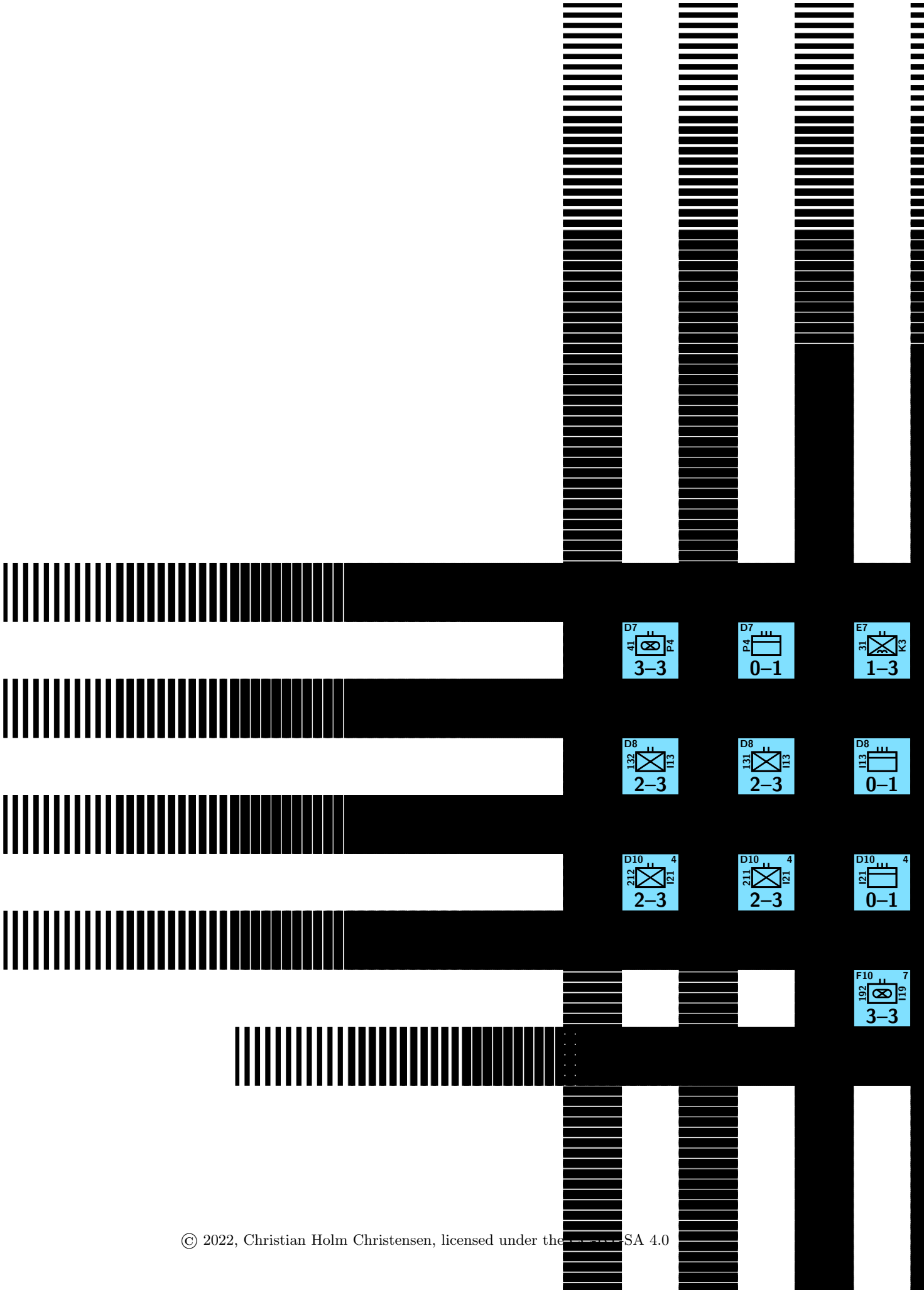


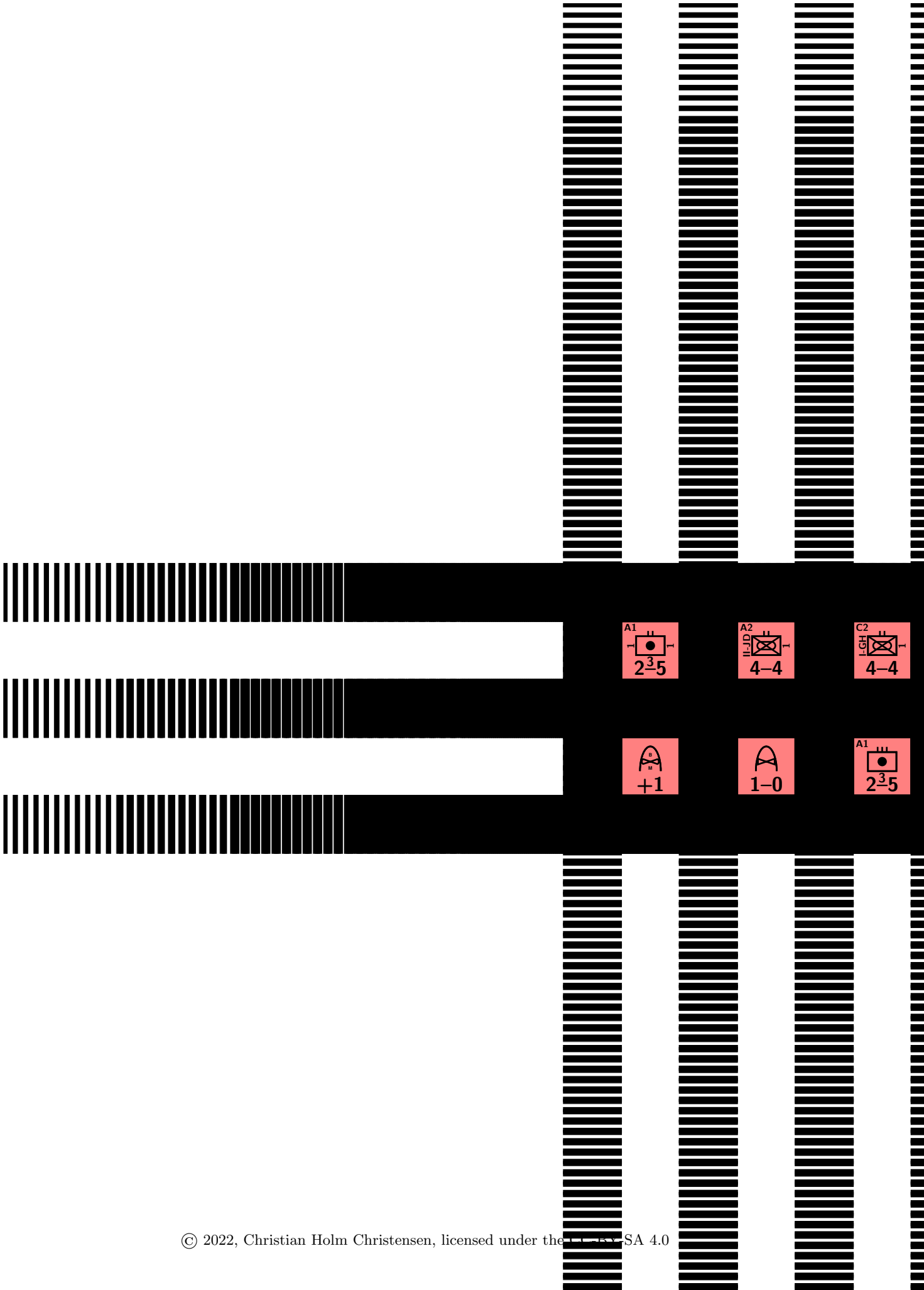
Die roll	Odds 1:3	1:2	1:1	2:1	3:1	4:1
1	AE	AR	AR	EX	DR	DR
2	AE	AR	EX	EX	DR	DR
3	AR	EX	EX	DR	DR	DE
4	AR	EX	EX	DR	DR	DE
5	EX	EX	DR	DR	DE	DE
5	EX	DR	DR	DE	DE	DE

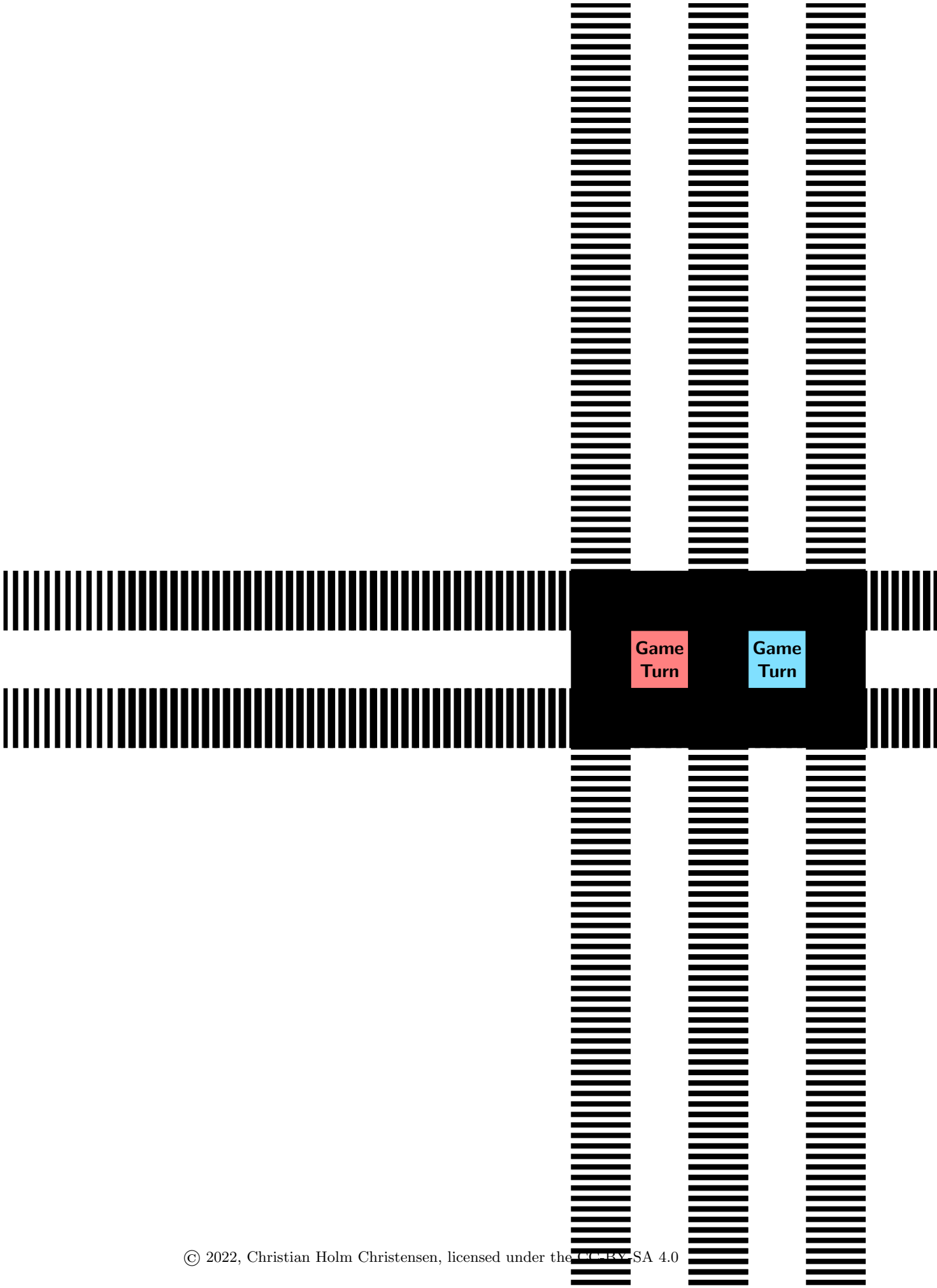
Terrain	MF	CF†
 Clear	1	—
 Woods	2	×2*
 Mountains	Stop	×2*
 Coastal	2	×2

† *Defender* modifier

* ⊠⊠ *only*, unless ⊡

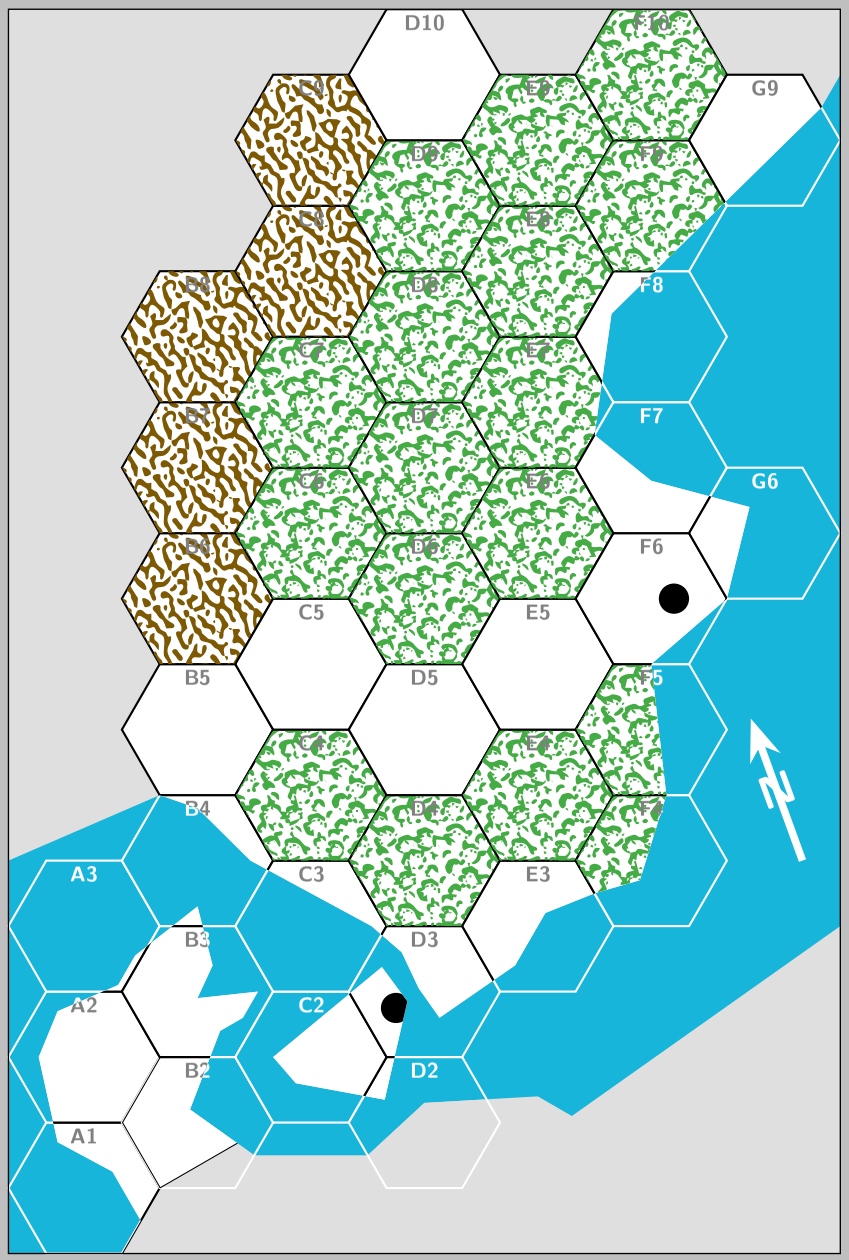






A Game

A tutorial in the wargame package

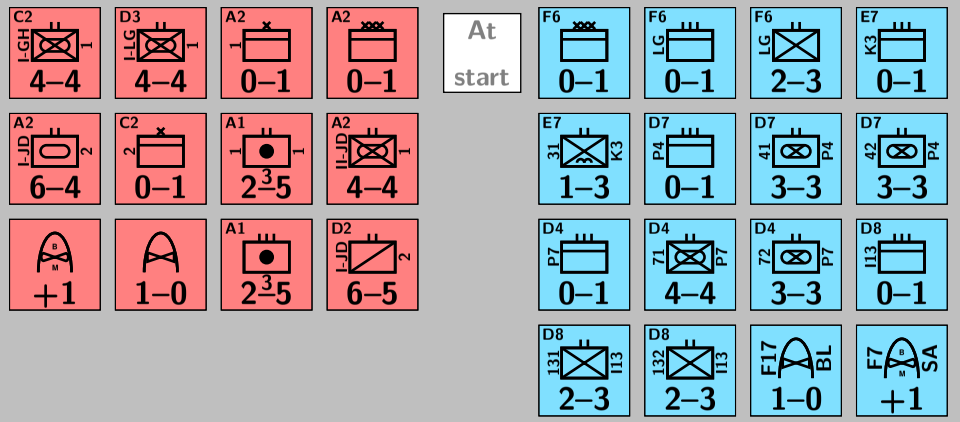


10 9 8 7 6 5 4 3 2 1

A Game

Order of Battle

Turn



Charts

Terrain	MF	CF†
Clear	1	—
Woods	2	×2*
Mountains	Stop	×2*
Coastal	2	×2

† *Defender* modifier

* ⊠⊠ *only*, unless ⊡

Die roll	Odds 1:3	1:2	1:1	2:1	3:1	4:1
1	AE	AR	AR	EX	DR	DR
2	AE	AR	EX	EX	DR	DR
3	AR	EX	EX	DR	DR	DE
4	AR	EX	EX	DR	DR	DE
5	EX	EX	DR	DR	DE	DE
5	EX	DR	DR	DE	DE	DE