The tucseating Package

v0.4.0 2025-07-23

Generation and visualization of seating plans

Matthias Werner[1]

https://github.com/tuc-osg/tucseating

This package allows for easy creation of seating charts, such as those required for examinations. Several automatic placement schemes are predefined, but custom, fine-grained seat assignments can also be specified. While the package was initially intended for internal use at Chemnitz University of Technology and contains predefined seating layouts for (some) of the rooms of TUC, on the one hand the room data can be easily expanded or replaced, and on the other hand rooms can also be created ad hoc.

Table of Contents

1. matthias.werner@informatik.tu-chemnitz.de

1 Introduction

To conduct exams, we occasionally require seating plans. Over time, we have accumulated several such plans in the form of TikZ-supported LaTeX files. Depending on the number of students in an exam—and how high we assess the risk of attempted cheating—we apply different placement schemes, which requires adapting the files accordingly. Moreover, we are sometimes assigned new rooms for which no seating plans exist yet.

This was the motivation for developing the tucseating package. It …

- enables quick and easy creation of seating plans;
- separates seat layout from the placement scheme;
- offers a set of standard placement schemes;
- includes a number of predefined rooms with seat layouts;
- allows for *ad hoc* creation of new rooms and placement schemes.

2 Dependencies

The tucseating package works only with LuaLaTeX and requires a sufficiently modern LaTeX version, at least from July 2022. It loads the following packages:

- etoolbox
- luacode
- tikz

These packages are available in all major TeX distributions and themselves depend on other packages. In particular, tikz loads the xcolor package, whose color definitions are also used by tucseating.

3 Seat Layout

The package is loaded as usual with `\usepackage[`⟨*options*⟩`]{tucseating}`

This allows the layout of the seats—i.e., their arrangement in the room— to be defined right away. Seat assignment is done later; see Section 4.

3.1 Class Options

The use of the option `room` distinguishes between two fundamental use cases:

`room` = {⟨*Room*⟩}

This option key should be used if the desired room is already predefined.

`room file` = {⟨*filename*⟩} Default: `rooms1`

The data for the predefined rooms is read from ⟨*filename*⟩`.tsr`. This key can be used when creating your own rooms, see Section 6.

If the room is not already known, it can also be created *ad hoc*, whereas several key–value options are required to describe the layout:

`shape` = `rectangle|arc` Default: `rectangle`

Specifies whether the seating layout is rectangular (`rectangle`) or curved (`arc`).[2]

`rows` = {⟨*number*⟩} (required)

`seats per row` = {⟨*number*⟩} (required)

Defines the number of seat rows and seats per row. The maximum number must always be specified here. For incomplete rows, seats will be removed later, but no additional seats can be added that lie outside the originally defined layout area. **Note:** Aisles between blocks of seats must be counted here as seats as well.

If you specify **both** the `room` key **and** one of the keys `rows` or `seats per row` in the class options, the result is undefined. A warning will be issued in that case.

All further class options define the visual representation of the room layout. They can also be set later using `\tucsConfig`; see Section 3.2.

`blackboard` = `true|false` Default: `false`

Draws a blackboard.

`seat distance` = {⟨*distance*⟩} Default: `2pt`

Distance between seats. This also determines the row spacing. If you want these two values to differ, you must use the following keys:

`seat neighbor distance` = {⟨*Distance*⟩} Default: `2pt`

`row distance` = {⟨*distance*⟩} Default: `2pt`

`rownumbers` = `none|left|right|both` Default: `none`

For rectangular layouts, defines whether to display row numbers on the left, right, or both sides. Counting starts at the front (blackboard side).
Note: For arc-shaped layouts, this function is currently not implemented.

`rownumber distance` = {⟨*distance*⟩} Default: `2pt`

Distance between the row number and the outermost seats, if `rownumbers` is not `none`.

`empty seat background color` = {⟨*color*⟩} Default: `lightgray!20`

`empty seat border color` = {⟨*color*⟩} Default: `lightgray`

2. For example, room A10.316 at TU Chemnitz uses an arc layout.

`assigned seat background color` = {⟨*color*⟩} Default: `lightgray!30`

`assigned seat border color` = {⟨*color*⟩} Default: `black`

Defines background and border colors for empty or assigned seats. The syntax of color names follows the specification of the xcolor package. If the seats should not be differentiated by color, the keys

`seat background color` = ⟨*color*⟩ (initially empty)

`seat border color` = ⟨*color*⟩ (initially empty)

can also be used.

`assigned seat label font` = {⟨*font command*⟩} Default: `\small`

`assigned seat label color` = ⟨*color*⟩ Default: `black`

Sets the size and color of the seat label.

3.2 Late Option Selection

`\tucsConfig`

This command can be used to set all keys mentioned in Section 3.1, except for `room`, `shape`, `rows`, and `seats per row`, outside of the `\documentclass` options.

3.3 Modifying the Seat Layout

Often, not all seats are present in every row. This is already taken into account when specifying a predefined room, but it may still be necessary to remove individual seats — for instance, if a folding seat is broken.

When creating a custom layout, such modifications are typically required. For this purpose, provides the following commands:

`\tucsRemoveSeatAt`{⟨*row*⟩}{⟨*seat number*⟩}

Removes seat ⟨*seat number*⟩ in row ⟨*row*⟩ from the layout. The values refer to the original layout and are unaffected by previously removed seats. If ⟨*seat number*⟩ is negative, counting starts from the right.

`\tucsRemoveSeats`{⟨*list*⟩}

Removes all seats listed in the comma-separated ⟨*list*⟩, where each entry is of the form {⟨*row*⟩,⟨*seat number*⟩}. As with `\tucsRemoveSeatAt`, row and seat numbers refer to the original layout. In the following example, the first three seats on the left and the first seat on the right in the first row are removed:

```
1  \tucsRemoveSeats{{1,1},{1,2},{1,3},{1,-1}}
```

`\tucsSetAisle`[⟨*start row*⟩-⟨*end row*⟩]{⟨*seat number*⟩}

Inserts an aisle at the position of ⟨*seat number*⟩ through all rows. Use the optional argument if the aisle should not span all rows.

For horizontal aisles (which split a seat block into front and back instead of left and right), please use \tucsRemoveSeats.

While this is also possible for vertical aisles, automatically generated seating schemes may treat aisles created via \tucsSetAisle differently than those created with \tucsRemoveSeats.

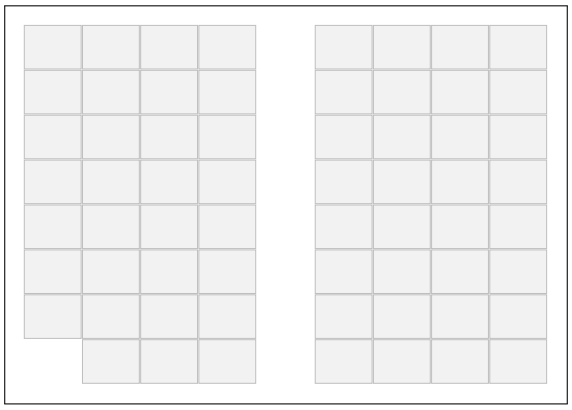
3.4 Output

\tucsDrawSeating[seat width=⟨*width*⟩, seat height=⟨*height*⟩]

Renders the seating plan. tucseating attempts to calculate the seat dimensions so that the available space for the complete plan is fully utilized. Since this may not always meet all needs, the optional argument allows the width and height of the seats on the plan to be adjusted individually.

The following example shows the code and the result for a simple seating layout with a central aisle and one missing seat in the first row.

```
\documentclass{scrartcl}
% To make better use of the paper, we use landscape format
% and reduce the margins
\usepackage[a4paper,landscape,inner=1cm,outer=1cm,top=1cm,bottom=1cm]{geometry}
\usepackage[shape=rectangle, rows = 8, seats per row=9]{tucseating}
\begin{document}
  \tucsSetAisle{5}
  \tucsRemoveSeatAt{1}{1}
  \tucsDrawSeating
\end{document}
```



4 Seat Assignment

4.1 Parameters for Constructing Assignment Schemes

A seating plan is only partially useful without indicating which seats are occupied. In tucseating, seat assignment is handled via assignment schemes. To define such a scheme, the command

\tucsSeatingScheme[⟨*key list*⟩]{⟨*name*⟩}
\tucsSeatingScheme*[⟨*key list*⟩]{⟨*pattern*⟩}

The mandatory argument, in the standard version of the command, is a name for a predefined seating scheme (see Section 4.2). In the starred version, a string of "`X`" and "`-`" is passed, describing (the beginning of) a row assignment pattern, where "`X`" denotes an occupied seat and "`-`" an empty one. If the string is shorter than the number of seats in a row, it will be repeated.

For example,

```
1 \tucsSeatingScheme*{X--}
```

marks every third seat as occupied.

The optional argument accepts a list of key–value pairs that control the remaining settings of the assignment scheme. These are especially important in the starred variant but may also be used in the standard one.

`row sep` = ⟨*number*⟩ Default: `2`
Specifies how many rows should lie between two occupied rows.

`start row` = ⟨*row*⟩
First row to be included in the assignment.

`end row` = ⟨*row*⟩
Last row to be included in the assignment.

`row restart after` = ⟨*number*⟩ (initially empty)
Resets the seat pattern after ⟨*number*⟩ rows. This allows for alternating row spacing; see the example in Section 5.3.

`aisle counts` = ⟨*number*⟩ Default: `1`
Specifies how many seats wide an aisle should be considered. This allows for wider aisles to be taken into account.

`aisle restarts scheme` = `true`|`false` Default: `false`
Restarts the scheme after an aisle. In that case, the `aisle counts` key has no effect.

`ignore aisle` = `true`|`false` Default: `false`
`ignore removed seats` = `true`|`false` Default: `false`
When determining seat numbers (used e.g. in labels; see Section 4.3), aisles and removed seats are normally counted. This ensures that seat numbers are consistent across rows (in rectangular

layouts). If one of these keys is set, aisles or removed seats are no longer counted. This is particularly useful in curved layouts.

`assigned seat label` = {⟨*format string*⟩} Default: `mD`

Specifies how assigned seats should be labeled. By default, this consists of the row number, a thin space, and a letter for the current occupied seat (e.g., "'3 c"'). A number of other formats can be configured; see Section 4.3 for details.

\tucsConfigScheme{⟨*key list*⟩}

Sets keys just like the optional argument of \tucsSeatingScheme, but does not assign seats.

Key values set by \tucsSeatingScheme or \tucsConfigScheme remain in effect until they are explicitly redefined.

4.2 Predefined Seating Schemes

The tucseating package defines a set of predefined seating schemes. Some of them also have alternative names.

`1x1`

Every seat is marked.
Alternative name: `all`

`2x2`

One empty seat and one empty row between two occupied ones.
Alternative name: `simple`

`2x2-`

One empty seat between two occupied ones. After a single empty row, *two* rows with occupied seats follow. This scheme allows the largest number of students in a room during an exam while still maintaining minimal lateral distance and enabling supervision to reach every student (from the front or the back).
Alternative name: `dense`

`2x3`

Occupied seats have two empty seats between them laterally, and one empty row between occupied rows. This is considered the preferred default scheme for exams in our group.
Alternative name: `sixpack`

`2x3-`

Rows are occupied as in `2x2-`, but the lateral spacing within an occupied row is two seats.

`2x4`

Occupied seats have three empty seats between them laterally, and one empty row between occupied rows.

`3x4`

Occupied seats have three empty seats between them laterally, and two empty rows between occupied rows.

4.3 Seat Labeling

The labeling of assigned seats can be done in various ways, controlled by assigning a format string to the key `assigned seat label`. Four counters are available for this purpose:

- absolute row: the row number in the seat layout
- occupied row: the number of the *assigned* row. Rows without any assigned seats are skipped.
- absolute seat number: the seat number in the seat layout. Whether removed seats are counted depends on the value of `ignore removed seats`.
- occupied seat number: the number of the *assigned* seats. Unassigned seats are skipped in this count.

Each of these counters can be formatted differently. For this, the format string uses format specifiers listed in the following table:

Counter	**Description**	**Rendered as…**				
		Arabic numeral	lowercase letter	uppercase letter	lower Roman numeral	upper Roman numeral
absolute row	*based on seat layout*	`m`	`a`	`A`	`y`	`Y`
occupied row	*based on assigned rows*	`r`	`b`	`B`	`i`	`I`
absolute seat number	*based on seat layout*	`n`	`c`	`C`	`x`	`X`
occupied seat number	*based on assigned seats*	`s`	`d`	`D`	`j`	`J`

Parts of the label that should not be interpreted as format specifiers must be wrapped in double braces: {{⟨*protected string*⟩}}.

For example,

```
1 \tucsSeatingScheme[assigned seat label=Y{{-}}D]{2x3}
```

produces the label "'III-B"' for the fourth seat (from the left) in the third row.

4.4 Seat List

Especially in the context of examinations, it is useful to generate a seat list, i.e., a tabular mapping between student and assigned seat. In its current version, tucseating does not create a LaTeX table, but it can support table generation (e.g., using LaTeX or a spreadsheet application).

`\tucsSeatingList[`⟨*input file*⟩`]{`⟨*output file*⟩`}`

\tucsSeatingList*[⟨*input file*⟩]{⟨*output file*⟩}
Creates a CSV file ⟨*output file*⟩ containing the labels of the assigned seats, one per line.

> **Note!** The label texts written to ⟨*output file*⟩ are extracted from the corresponding LaTeX boxes, and only printable characters are included. When using this function, it's therefore recommended to avoid excessive TeX-level formatting in the `assigned seat label` format string.

In the starred variant, each line is prefixed with the absolute coordinates (row, seat number, relative to the layout), comma-separated, before the label.

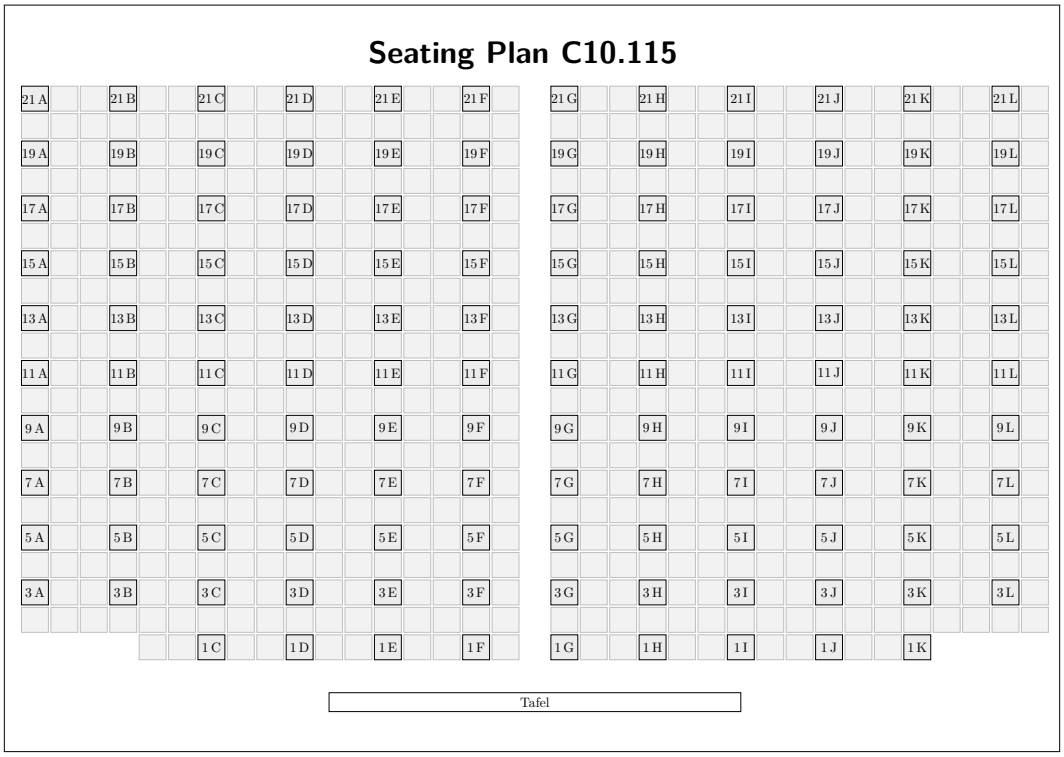
If an input file is specified, its content is prepended line-by-line to the generated lines. For example, the input file could contain names and/or student IDs, which will then be matched to the assigned seats in the output. If there are fewer entries in ⟨*input file*⟩ than assigned seats, the corresponding fields in the output will be left empty. If, on the other hand, there are more entries than available seats, tucseating will report which entries could not be assigned a seat.

5 Examples

To demonstrate the behavior of tucseating, a few usage examples are documented here.

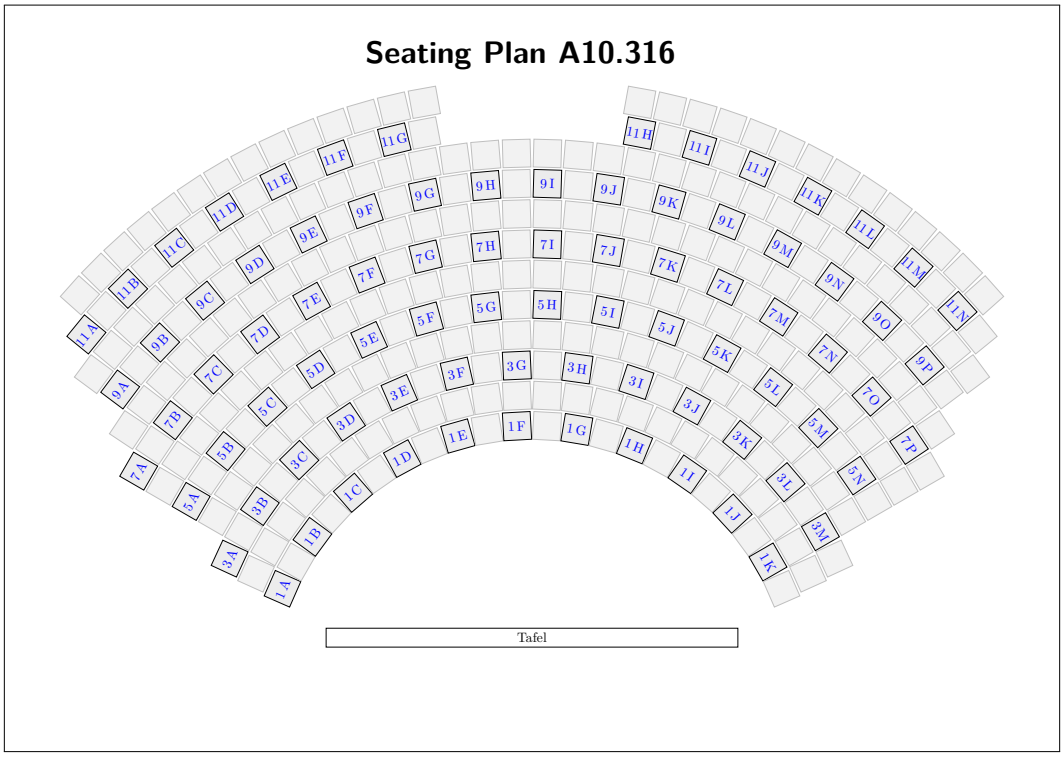
5.1 Predefined Room, Rectangular

```
\documentclass{scrartcl}
  % To make better use of the paper, we use landscape format
  % and reduce the margins
  \usepackage[a4paper,landscape,inner=10pt,outer=10pt,top=1cm,bottom=1cm]{
  geometry}
  \usepackage[
    room=C10.115,
    blackboard
  ]{tucseating}

\begin{document}
  % Default title takes up too much space.
  \centering\textbf{\Huge\sffamily Seating Plan C10.115}\bigskip
  \tucsSeatingScheme{sixpack}
  \tucsDrawSeating
\end{document}
```



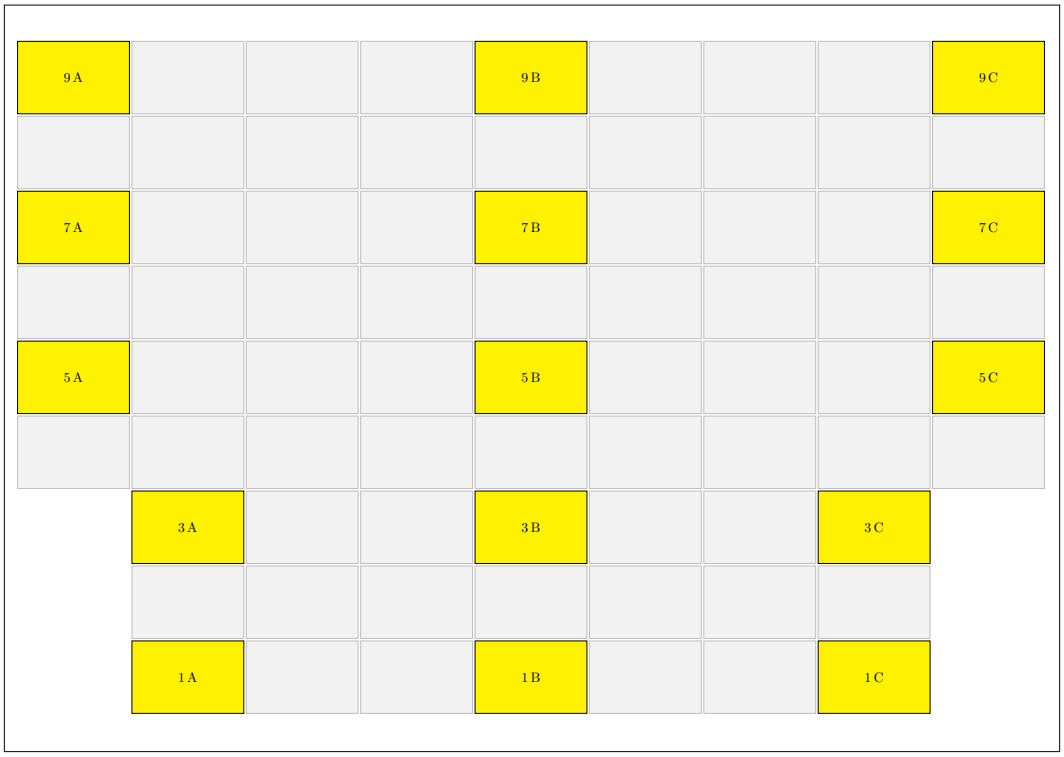
5.2 Predefined Room, Curved

```
\documentclass{scrartcl}
  % To make better use of the paper, we use landscape format
  % and reduce the margins
  \usepackage[a4paper,landscape,inner=10pt,outer=10pt,top=1cm,bottom=1cm]{geometry}
  \usepackage[
    room=A10.316,
    blackboard,
    assigned seat label color=blue
  ]{tucseating}

 \begin{document}
   % Default title takes up too much space.
   \centering\textbf{\Huge\sffamily Seating Plan A10.316}\bigskip
   \tucsSeatingScheme[ignore removed seats]{2x2}
   \tucsDrawSeating
 \end{document}
```



5.3 Custom Layout and Seating Scheme

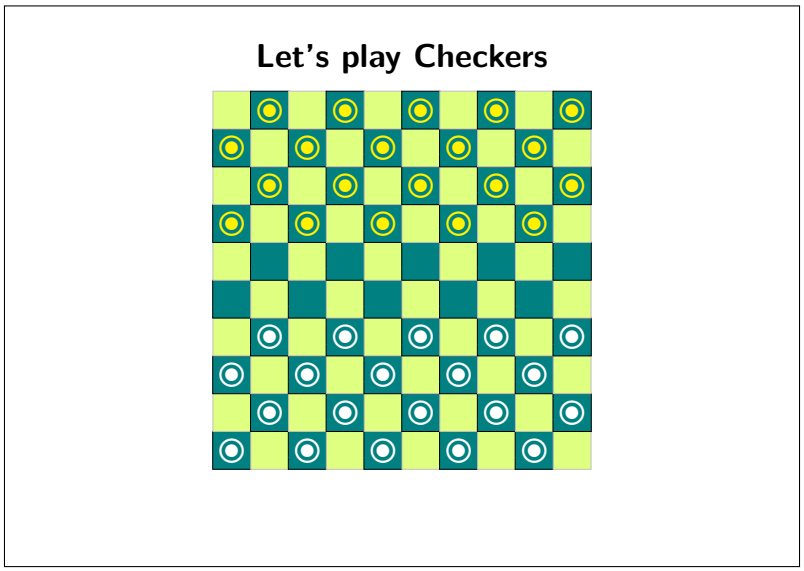
```
\documentclass{scrartcl}
  % To make better use of the paper, we use landscape format
  % and reduce the margins
  \usepackage[a4paper,landscape,inner=10pt,outer=10pt,top=1cm,bottom=1cm]{geometry}
  \usepackage[
    shape=rectangle, rows=9, seats per row=9,
    assigned seat background color=yellow
  ]{tucseating}

  \begin{document}
    \tucsRemoveSeats{{1,1},{1,-1},{2,1},{2,-1},{3,1},{3,-1}}

    \tucsSeatingScheme[ignore removed seats,end row=3]{2x3}
    \tucsSeatingScheme[start row=5, end row=10]{2x4}
    \tucsDrawSeating
\end{document}
```



5.4 Checkers

```
\documentclass{scrartcl}
  \usepackage[a5paper,landscape,top=1cm,bottom=1cm]{geometry}
  \usepackage[shape=rectangle, rows=10,seats per row=10,
     assigned seat label color=blue, assigned seat background color=teal,
     empty seat background color=lime!50,
     seat distance=0pt]{tucseating}
  \usepackage{stix}
  \begin{document}
    \centering\textbf{\Huge\sffamily Let's play Checkers}\bigskip

    \tucsConfigScheme{assigned seat label={}}
    \tucsSeatingScheme*[start row=5, end row=5]{X-}
    \tucsSeatingScheme*[start row=6, end row=6]{-X}
    \tucsConfigScheme{assigned seat label font=\Huge, assigned seat label={{{\
  textcolor{white}{$\circledbullet$}}}}}
    \tucsSeatingScheme*[start row=1, end row=3,row sep=2]{X-}
    \tucsSeatingScheme*[start row=2, end row=4]{-X}
    \tucsConfigScheme{assigned seat label={{{\textcolor{yellow}{$\circledbullet
  $}}}}}
```

```
18      \tucsSeatingScheme*[start row=7, end row=9]{X-}
19      \tucsSeatingScheme*[start row=8, end row=10]{-X}
20
21      \tucsDrawSeating[seat width=1cm, seat height=1cm]
22  \end{document}
```



6 Declaration of New Rooms

In the current version, the tucseatingpackage only contains a certain number of rooms with their seating layouts. Users will therefore often have to create their own layouts. In addition to the option for ad-hoc layout creation, as described in the Section 3, it is also possible to extend the package itself and thus create layouts for new rooms, which can then be easily accessed via the `room` key. New room layouts can be integrated in two ways:

- The `room1.tsr` file belonging to the tucseatingpackage is extended.
- A separate `.tsr` file is created and integrated using the `layout file` option.

Regardless of which approach is chosen, two commands can be used in `.tsr` files:

`\tucsDeclareRoom`{⟨*room name*⟩}{⟨*key list*⟩}
A new room layout is created for the room ⟨*room name*⟩. The key list *must* contain the three keys

`shape` = `rectangle|arc` (required)

`rows` = ⟨*number of rows*⟩ (required)

`seats per row` = ⟨*seats per row*⟩ (required)
These keys have the same meaning as the keys of the same name from Section 3. This information about the base layout *must* be followed by the key

`init` (required)

The layout can then be adjusted using

`aisle` = ⟨*seat number*⟩
`remove` = {⟨*list*⟩}
These two keys function analogously to `\tucsSetAisle` and `\tucsRemoveSeats`, see Section 3.3.

```
\tucsDeclareRoom{C10.115}{
  shape=rectangle,
  rows=21,
  seats per row=35,
  init,
  aisle=18,
  remove={{1,1},{1,2},{1,3},{1,4},{1,-1},{1,-2},{1,-3},{1,-4}}
}
```

`\tucsAliasRoom`{⟨*alias*⟩}{⟨*room*⟩}
Sets ⟨*alias*⟩ as a replacement name for ⟨*room*⟩.

7 Limitations and Bugs

Even though the tucseating package is in practical use, at least in our group, it should be considered experimental. The `0` in the major version number indicates this. In particular, it means that the API is still subject to change.

The package is subject to some limitations. Whether these will be fixed in future versions is not yet known. The package is designed for creating separate seating plans and therefore cannot be used to display multiple seating plans in one document. Likewise, seating layouts with staggered rows cannot be displayed.

The number of predefined rooms is currently relatively small. There are plans to include additional rooms from TU Chemnitz in future patches. Your collaboration by sending information about seating layouts or corresponding patches of `rooms1.tsr` is expressly welcome.

There are currently no known bugs, but some will certainly exist. Please use GitHub for bug reports or pull requests: `https://github.com/tuc-osg/tucseating`.

8 License