User guide for the PeRseus project

Jiri Munich

January 26, 2020

This is a user guide for the immediate use of the future **PeRseus** package.

Dependencies

Except for a color scheme imported in the plotting functions, the whole project is written in base R, so no extra package installations should be necessary.

```
## Load packages for plotting (optional)
library("RColorBrewer")
library("plotrix")
```

Setup

To initiate the project, source the following files in prescribed order:

```
run <- c("Basic/Basic_functions.R", "Basic/Dominate.R",
         "Robots/Naive.R", "Robots/Dominate-K.R",
         "Basic/K-1.R", "Robots/K-level.R",
         "Robots/Nash.R", "Robots/Optimist.R",
         "Robots/Pessimist.R", "Robots/Altruist.R",
         "Basic/Get_data_functions.R", "Basic/Plot_functions.R")

invisible(lapply(run, source))
```

To proceed with the replication, continue going through **model_jags_implementation.R**. What follows is a desription of functions written for this project.

Sorry about the uncivilized state. Don't hesitate to contact me with more questions.

Folder structure

Right now, the project script is divided into two folders. Folder **Basic** contains some elementary functions used throughout the project. Folder **Robots** contains implementations of used decision rules. In the following two sections, I will explain the content of these two folders. Five files can be found in the **Basic** folder. **Basic_functions.R**, **Dominate.R** and **K-1.R** contain functions used for algorithms in the **Robots** folder. **Get_data_functions.R** contains some functions used for the retrieval of data and **Plot_functions.R** contains functions for the visualization of generated data. The folder **Robots** contains implementation of the decision rule algorithms.

The intended order of use is following: **Basic_functions.R**, **Dominate.R** and **K-1.R** are used to create instruments for constructing the robots. Then, the **Robots** folder is used to produce the robots and finally, **Get_data_functions.R** are used to retrieve data using the robots. **Plot_functions.R** can be used to produce visualizations.

The files will be explained in the afforementioned order.

Basic_functions, Dominate and K-1

Basic_functions

File **Basic_functions.R** contains elementary operations used within decision rules.

my_max(); my_min

Find the index of a maximum/minimum of a vector of numeric values. If there are multiple maxima/minima, the functions select on at chance. The functions return a list containing the indices and a flag, noting whether there were multiple identified maxima/minima.

```
value_vec <- c(1,2,3,2,1)
my_max(value_vec)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] FALSE
```

```
my_min(value_vec)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] TRUE
```

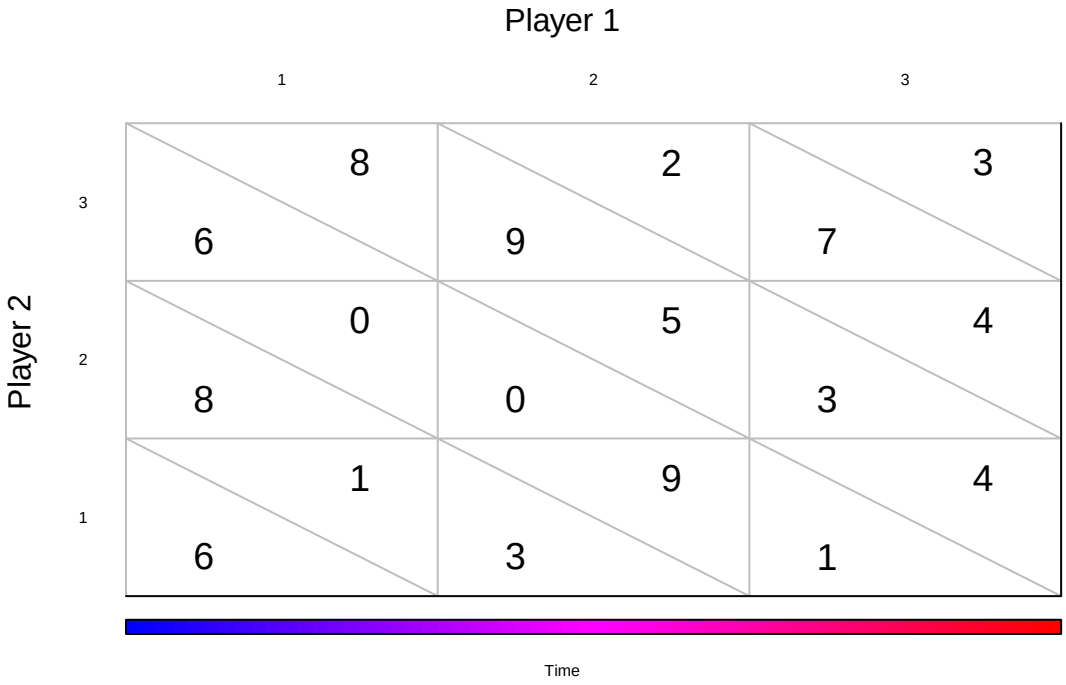
generate_game()

Generates a game with random payoffs with p1 choices available to player 1, p2 choices available to player 2 and payoff values coming from vector y. Player 1 choices are represented on the horizontal axis. Player 1 payoffs are represented by the even columns. The payoff matrix can be seen in the plot with inverted vertical axis.

```
# Generate a 3x3 game with payoffs ranging from 0 to 9
game <- generate_game(3,3,0:9)
print(game)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    6    1    3    9    1    4
## [2,]    8    0    0    5    3    4
## [3,]    6    8    9    2    7    3
```

```
e_o <- search_k.level(game,2)
e_o$eye_movement <- list(c(10,10),c(20,20))
plot_eye(e_o)
```



game_own(); game_other()

Splits the game into a matrix of own/other player's payoffs.

```
print(game)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    6    1    3    9    1    4
## [2,]    8    0    0    5    3    4
## [3,]    6    8    9    2    7    3
```

```
# Even columns represent own payoffs
game_own(game)
```

```
##      [,1] [,2] [,3]
## [1,]    1    9    4
## [2,]    0    5    4
## [3,]    8    2    3
```

```
game_other(game)
```

```
##      [,1] [,2] [,3]
## [1,]    6    3    1
## [2,]    8    0    3
## [3,]    6    9    7
```

transpose_game()

Transposes the game, so it is seen from the other player's perspective

```
## Look at the game from the perspective of player 2
t_game <- transpose_game(game)
print(t_game)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    6    0    8    8    6
## [2,]    9    3    5    0    2    9
## [3,]    4    1    4    3    3    7
```

```
## See the game as player 1 again by transposing the transposition
t_t_game <- transpose_game(t_game)
print(t_t_game)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    6    1    3    9    1    4
## [2,]    8    0    0    5    3    4
## [3,]    6    8    9    2    7    3
```

```
## Transposed transposition equals the original game
all.equal(game, t_t_game)
```

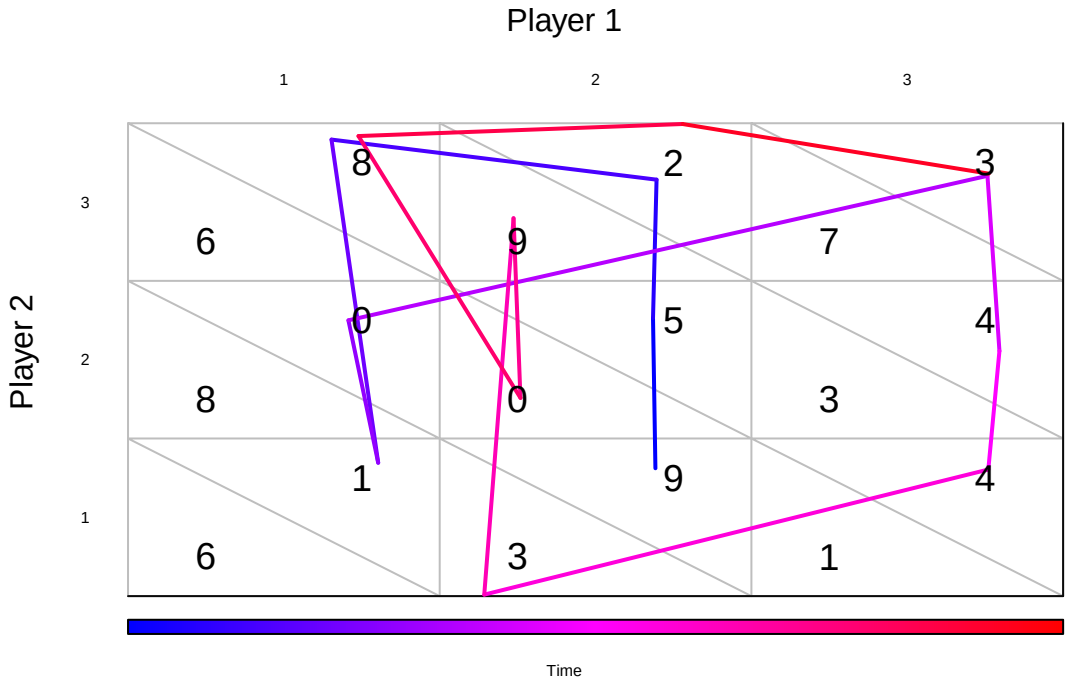
```
## [1] TRUE
```

transpose_eye()

Takes eye_movement coordinates and transposes the coordinates, so they can be mapped on a transposed game

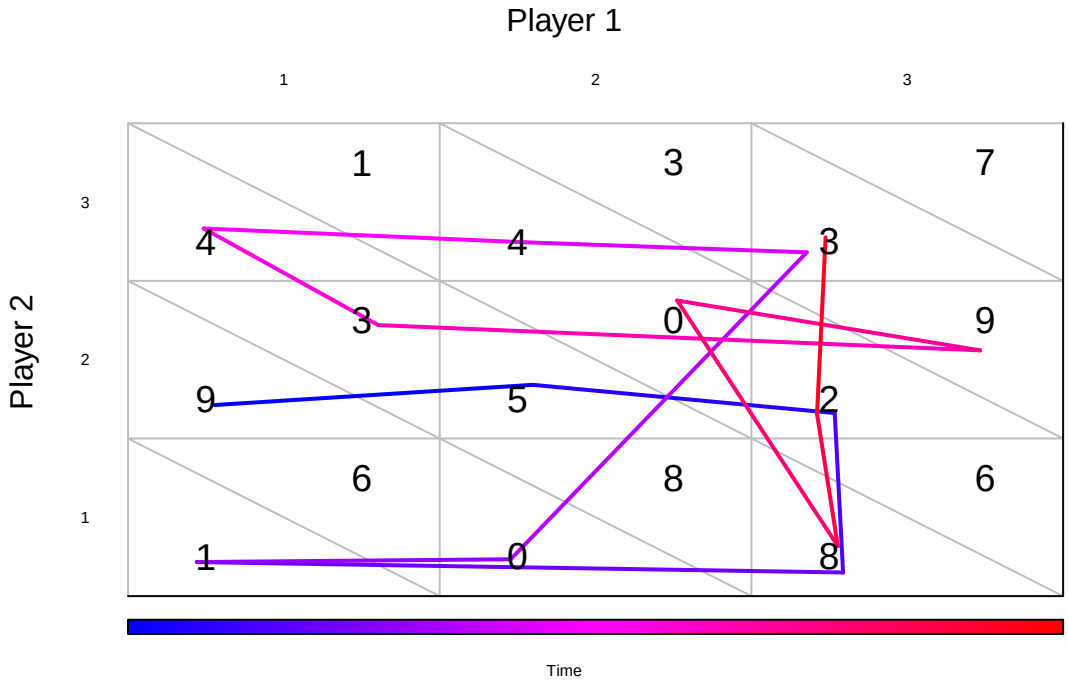
```
## In this example, e_o is an eye movement object, containing the original game payoff
## structure and a list of coordinates containing eye movement.
## e_o can be plotted

e_o <- search_k.level(game,2)
plot_eye(e_o)
```




```
## Transposing the game matrix and the eye movement coordinates,
## plotting e_o produces the same plot on a transposed game

e_o$eye_movement <- transpose_eye(e_o$eye_movement, e_o$game)$eye_data
e_o$game <- transpose_game(game)
plot_eye(e_o)
```



search_maximum()

Given a set of coordinates and payoff values in the coordinates, the function reproduces search for the highest value in the set by looking up payoff values, storing them in memory and droping dominated values.

```
meta_memory <- c(2,4,6,8,10)
loc_meta_memory <- list(c(1,1),c(2,2),c(3,3),c(4,4),c(4,5))

search_maximum(meta_memory = meta_memory, loc_meta_memory = loc_meta_memory)
```

```
## $eye_choice
## [1] 5
##
## $eye_movement
## $eye_movement[[1]]
## [1] 4 5
##
## $eye_movement[[2]]
## [1] 2 2
##
## $eye_movement[[3]]
## [1] 2 2
##
## $eye_movement[[4]]
## [1] 4 4
##
## $eye_movement[[5]]
```

```
## [1] 4 4
##
## $eye_movement[[6]]
## [1] 1 1
##
## $eye_movement[[7]]
## [1] 1 1
##
## $eye_movement[[8]]
## [1] 3 3
```

Dominate and K-1

domination(); search_k.1()

Perform one itteration of removing dominated choices/performing k-level reasoning. Operate as the robots and will be explained in the **Robots** section.

Robots

Functions in this folder are the decision rule algorithms. All start with a prefix search_ and continue with the given algorithm. They use a **game** as an input and return an **eye-object**. K-level and domination are itterative decision rules and therefore contain a **k** parameter indicating the level of reasoning.

- **game** is a matrix of payoff values as described for generate_game() in the previous section.
- **eye-object** is a list consisting of:
- **an_choice**: is an index of the choice selected by the algorithm, using a quasianalytical solution. Besides performing an information search through the game matrix, a simpler implementation of each decision rule was used, applying simple functions. The purpose of **an_choice** is to validate the decision rule implementation. Each choice resulting from an information search can be compared to the choice done by simple matrix operations applying the same rule. If **flag** is TRUE, the **an_choice** and **eye_choice** are allowed to differ, as will be explained later.
- **eye_choice** is an index of the choice selected by the algorithm after performing its information search.
- **eye_movement** is an ordered list of coordinates recording all locations visited by the algorithm.
- **flag** is used to monitor situations where more solutions were available. In such case, the algorithms decides randomly between equivalently valued choices (and in such situation, **an_choice** and **eye_choice** may differ). Flag can be also used to find games that do not offer multiple solutions given a rule.
- **game** is the original game used as an input. It is stored in the **eye_object** for future reference.

```
## In this example, an altruist information search is performed, returning an eye_object (printed)
alt_eye_object<- search_altruist(game)
print(alt_eye_object)
```

```
## $an_choice
## [1] 1
##
## $eye_choice
## [1] 1
##
## $eye_movement
## $eye_movement[[1]]
## [1] 3 6
##
```

```
## $eye_movement[[2]]
## [1] 1 6
##
## $eye_movement[[3]]
## [1] 3 5
##
## $eye_movement[[4]]
## [1] 1 5
##
## $eye_movement[[5]]
## [1] 2 6
##
## $eye_movement[[6]]
## [1] 2 5
##
## $eye_movement[[7]]
## [1] 1 3
##
## $eye_movement[[8]]
## [1] 3 3
##
## $eye_movement[[9]]
## [1] 2 3
##
## $eye_movement[[10]]
## [1] 2 4
##
## $eye_movement[[11]]
## [1] 1 4
##
## $eye_movement[[12]]
## [1] 3 4
##
## $eye_movement[[13]]
## [1] 3 2
##
## $eye_movement[[14]]
## [1] 1 1
##
## $eye_movement[[15]]
## [1] 1 2
##
## $eye_movement[[16]]
## [1] 2 1
##
## $eye_movement[[17]]
## [1] 3 1
##
## $eye_movement[[18]]
## [1] 2 2
##
##
## $flag
## [1] FALSE
```

```
##
## $game
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    6    1    3    9    1    4
## [2,]    8    0    0    5    3    4
## [3,]    6    8    9    2    7    3
```

Get_data_functions

Having produced eye_objects, data can be classified into (meta-)transition types and used in further analysis.

get_transitions()

Can be provided either with **eye_movement** type data (a list of vectors with two dimensional coordinates), or an entire **eye_object** (type of data must be specified in the function parameters). Uses parameter **n** indicating the order of meta-transitions (defaults at 1).

Returns a list containing: * **transitions** a character vector with simple transition types. * **meta-transitions** a character vector containing the requested type of meta-transitions.

```
## Get transitions from eye movement coordinates
get_transitions(eye_data = alt_eye_object$eye_movement)
```

```
## $transitions
##  [1] "own-within"    "skip"          "other-between" "skip"
##  [5] "intracell"     "other-between" "other-between" "other-between"
##  [9] "intracell"     "own-within"    "own-within"    "own-between"
## [13] "skip"          "intracell"     "skip"          "other-between"
## [17] "skip"
##
## $meta_transitions
##  [1] "own-within_skip"             "skip_other-between"
##  [3] "other-between_skip"          "skip_intracell"
##  [5] "intracell_other-between"     "other-between_other-between"
##  [7] "other-between_other-between" "other-between_intracell"
##  [9] "intracell_own-within"        "own-within_own-within"
## [11] "own-within_own-between"      "own-between_skip"
## [13] "skip_intracell"              "intracell_skip"
## [15] "skip_other-between"          "other-between_skip"
```

```
## Get transitions from an eye object
get_transitions(eye_object = alt_eye_object, is.object = TRUE)
```

```
## $transitions
##  [1] "own-within"    "skip"          "other-between" "skip"
##  [5] "intracell"     "other-between" "other-between" "other-between"
##  [9] "intracell"     "own-within"    "own-within"    "own-between"
## [13] "skip"          "intracell"     "skip"          "other-between"
## [17] "skip"
##
## $meta_transitions
##  [1] "own-within_skip"             "skip_other-between"
##  [3] "other-between_skip"          "skip_intracell"
##  [5] "intracell_other-between"     "other-between_other-between"
##  [7] "other-between_other-between" "other-between_intracell"
##  [9] "intracell_own-within"        "own-within_own-within"
## [11] "own-within_own-between"      "own-between_skip"
```

```
## [13] "skip_intracell"              "intracell_skip"
## [15] "skip_other-between"          "other-between_skip"
```

```
## Get meta-transitions of the second order
get_transitions(eye_object = alt_eye_object, is.object = TRUE, n = 2)
```

```
## $transitions
##  [1] "own-within"    "skip"          "other-between" "skip"
##  [5] "intracell"     "other-between" "other-between" "other-between"
##  [9] "intracell"     "own-within"    "own-within"    "own-between"
## [13] "skip"          "intracell"     "skip"          "other-between"
## [17] "skip"
##
## $meta_transitions
##  [1] "own-within_skip_other-between"
##  [2] "skip_other-between_skip"
##  [3] "other-between_skip_intracell"
##  [4] "skip_intracell_other-between"
##  [5] "intracell_other-between_other-between"
##  [6] "other-between_other-between_other-between"
##  [7] "other-between_other-between_intracell"
##  [8] "other-between_intracell_own-within"
##  [9] "intracell_own-within_own-within"
## [10] "own-within_own-within_own-between"
## [11] "own-within_own-between_skip"
## [12] "own-between_skip_intracell"
## [13] "skip_intracell_skip"
## [14] "intracell_skip_other-between"
## [15] "skip_other-between_skip"
```

initiate_meta_transitions()

A function preparing a data frame for the collection of observations, intended for use by other functions. Contains one parameter, indicating the meta-transition order

```
init <- initiate_meta_transitions(n = 1)
```

write_eye_data()

Turns a list of lists of eye coordinates (**eye_movement** object) into a data frame with frequencies of meta-transitions of the requested level.

```
eye_list <- list(
  alt_eye_object$eye_movement,
  alt_eye_object$eye_movement
)

write_eye_data(eye_list, n=1)
```

```
##   intracell_intracell intracell_own-within intracell_own-between
## 1                   0                    1                     0
## 2                   0                    1                     0
##   intracell_other-within intracell_other-between intracell_skip
## 1                      0                       1              1
## 2                      0                       1              1
##   own-within_intracell own-within_own-within own-within_own-between
```

```
## 1                   0                    1                     1
## 2                   0                    1                     1
##   own-within_other-within own-within_other-between own-within_skip
## 1                       0                       0              1
## 2                       0                       0              1
##   own-between_intracell own-between_own-within own-between_own-between
## 1                     0                      0                       0
## 2                     0                      0                       0
##   own-between_other-within own-between_other-between own-between_skip
## 1                        0                         0                1
## 2                        0                         0                1
##   other-within_intracell other-within_own-within other-within_own-between
## 1                      0                       0                        0
## 2                      0                       0                        0
##   other-within_other-within other-within_other-between other-within_skip
## 1                         0                          0                 0
## 2                         0                          0                 0
##   other-between_intracell other-between_own-within other-between_own-between
## 1                       1                       0                         0
## 2                       1                       0                         0
##   other-between_other-within other-between_other-between other-between_skip
## 1                          0                           2                  2
## 2                          0                           2                  2
##   skip_intracell skip_own-within skip_own-between skip_other-within
## 1              2               0                0                 0
## 2              2               0                0                 0
##   skip_other-between skip_skip
## 1                  2         0
## 2                  2         0
```

simulate_search()

Produces a number of searches for a number of games, returning transition and meta-transition data.

Parameters are: * **games** the number of games to be randomly generated and used * **repetitions** indicating the number of times each game is played by every algorithm * **n** indicating the order of meta-transitions

Returns a list with: * **transition_data** a data frame with transition counts * **meta_transition_data** a data frame with meta-transition counts of the requested order * **games** a list with all games used getting the data

```
searches <- simulate_search(games = 2 , repetitions = 2, n = 1)
head(searches$meta_transition_data)
```

```
##   game      rule intracell_intracell intracell_own-within intracell_own-between
## 1    1  Optimist                   0                    0                     0
## 2    1 Pessimist                   0                    0                     0
## 3    1  Altruist                   0                    0                     0
## 4    1     Naive                   0                    0                     0
## 5    1       K-1                   0                    0                     0
## 6    1       K-2                   0                    0                     0
##   intracell_other-within intracell_other-between intracell_skip
## 1                      0                       0             0
## 2                      0                       0             0
## 3                      0                       0             2
## 4                      0                       0             0
```

```
## 5                     0                     0             0
## 6                     0                     1             0
##   own-within_intracell own-within_own-within own-within_own-between
## 1                    0                     4                     3
## 2                    0                     3                     4
## 3                    0                     0                     0
## 4                    0                     3                     2
## 5                    0                     0                     0
## 6                    1                     3                     2
##   own-within_other-within own-within_other-between own-within_skip
## 1                       0                       0              0
## 2                       0                       0              0
## 3                       0                       0              2
## 4                       0                       0              0
## 5                       0                       0              0
## 6                       0                       0              0
##   own-between_intracell own-between_own-within own-between_own-between
## 1                     0                      2                       1
## 2                     0                      3                       0
## 3                     0                      0                       0
## 4                     0                      2                       0
## 5                     0                      0                       1
## 6                     0                      2                       1
##   own-between_other-within own-between_other-between own-between_skip
## 1                        0                         0                0
## 2                        0                         0                0
## 3                        0                         0                0
## 4                        0                         0                0
## 5                        0                         0                0
## 6                        0                         0                0
##   other-within_intracell other-within_own-within other-within_own-between
## 1                      0                       0                        0
## 2                      0                       0                        0
## 3                      0                       0                        0
## 4                      0                       0                        0
## 5                      0                       0                        0
## 6                      0                       0                        0
##   other-within_other-within other-within_other-between other-within_skip
## 1                         0                          0                0
## 2                         0                          0                0
## 3                         0                          0                0
## 4                         0                          0                0
## 5                         3                          2                1
## 6                         0                          0                0
##   other-between_intracell other-between_own-within other-between_own-between
## 1                       0                        0                         0
## 2                       0                        0                         0
## 3                       0                        0                         0
## 4                       0                        0                         0
## 5                       0                        0                         0
## 6                       0                        0                         0
##   other-between_other-within other-between_other-between other-between_skip
## 1                          0                           0                  0
## 2                          0                           0                  0
```

```
## 3                         0                         0                3
## 4                         0                         0                0
## 5                         2                         0                0
## 6                         0                         1                1
##   skip_intracell skip_own-within skip_own-between skip_other-within
## 1              0               0                0                 0
## 2              0               0                0                 0
## 3              2               3                0                 0
## 4              0               0                0                 0
## 5              0               0                1                 0
## 6              0               0                1                 0
##   skip_other-between skip_skip
## 1                  0         0
## 2                  0         0
## 3                  2         2
## 4                  0         0
## 5                  0         0
## 6                  0         0
```

impossible_meta_transitions(); add_postion()

impossible_meta_transitions() returns a character vector of impossible meta-transitions of the requested order by performing every possible meta-transition of the requested order. **add_postion()** is a function written with a typo in its name. It is used to create all possible steps from a given list of position coordinates.

```
impossible_meta_transitions(3,3,1)
```

```
## [1] "own-within_other-within"   "own-within_other-between"
## [3] "own-between_other-within"  "own-between_other-between"
## [5] "other-within_own-within"   "other-within_own-between"
## [7] "other-between_own-within"  "other-between_own-between"
```

chance_meta_transition_probabilities()

Returns a list o probabilities of any level meta-transition in a matrix of a given size occuring by chance. The chance is computed from the proportion of n-long walks of a given type to all possible n-long walks.

```
chance_meta_transition_probabilities(p1 = 3, p2 = 3, n = 1)
```

```
##         intracell_intracell        intracell_own-within
##                 0.006920415                 0.006920415
##       intracell_own-between      intracell_other-within
##                 0.020761246                 0.006920415
##     intracell_other-between              intracell_skip
##                 0.020761246                 0.055363322
##        own-within_intracell       own-within_own-within
##                 0.003460208                 0.006920415
##      own-within_own-between     own-within_other-within
##                 0.020761246                 0.000000000
##    own-within_other-between             own-within_skip
##                 0.000000000                 0.027681661
##       own-between_intracell      own-between_own-within
##                 0.010380623                 0.020761246
##     own-between_own-between    own-between_other-within
##                 0.062283737                 0.000000000
##   own-between_other-between            own-between_skip
```

```
##                 0.000000000                 0.083044983
##      other-within_intracell     other-within_own-within
##                 0.003460208                 0.000000000
##    other-within_own-between   other-within_other-within
##                 0.000000000                 0.006920415
##  other-within_other-between           other-within_skip
##                 0.020761246                 0.027681661
##     other-between_intracell    other-between_own-within
##                 0.010380623                 0.000000000
##   other-between_own-between  other-between_other-within
##                 0.000000000                 0.020761246
## other-between_other-between          other-between_skip
##                 0.062283737                 0.083044983
##              skip_intracell             skip_own-within
##                 0.024221453                 0.024221453
##            skip_own-between           skip_other-within
##                 0.072664360                 0.024221453
##          skip_other-between                   skip_skip
##                 0.072664360                 0.193771626
```

simulate_search_get_data()

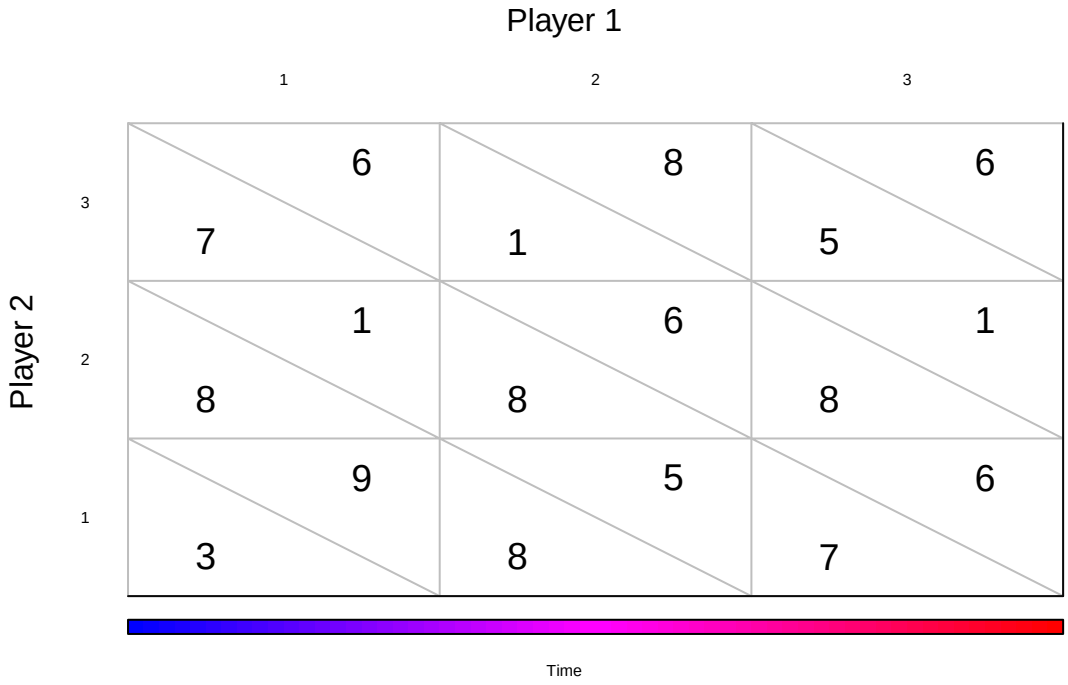
Is used for aggregating simulated data in order to determine meta-transition typical for decision rules.

Plot_functions

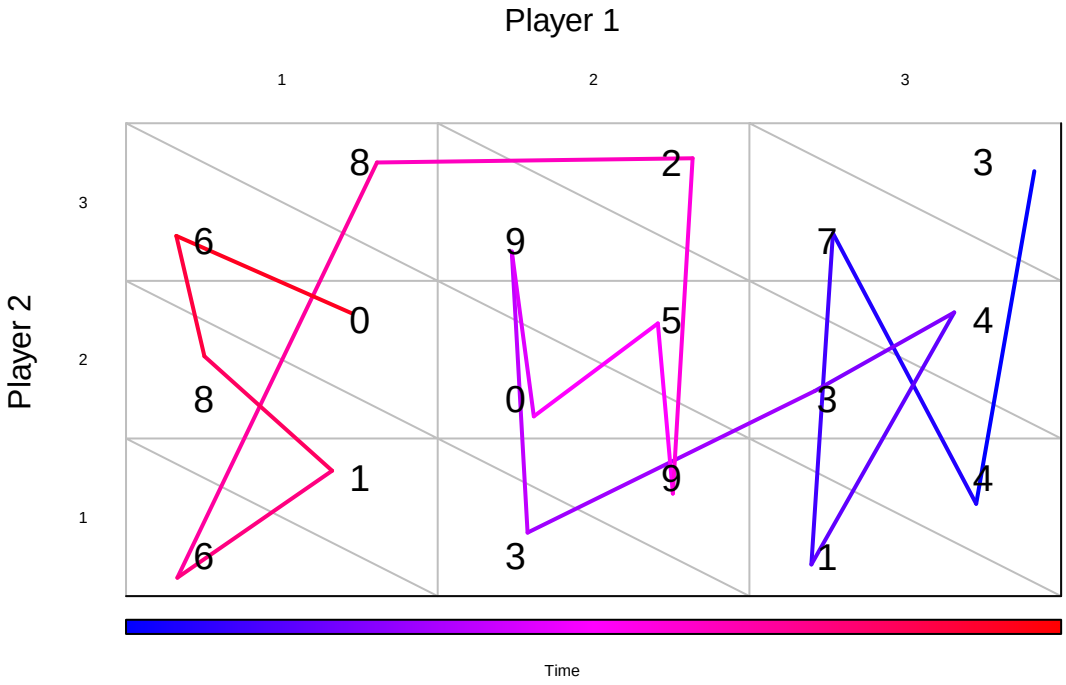
plot_eye()

Uses an **eye_object** to plot information search. Some jitter can be added to the coordinates to distinguish overlapping transitions. When used empty, the function produces a plot with a random game.

```
plot_eye()
```




```
plot_eye(alt_eye_object)
```



animate_search() and animate_search_gif()

animate_search() produces an animation by plotting information search of an **eye_object** in itterations. The gif version instead saves frames into a folder. The images can be used to produce a gif animation.