Distributing fixed resources over many items. An experimental interface*

Alessandra Casella† and Olivier Compte‡

May 2, 2026

Abstract

We propose an experimental interface designed to elicit the distribution of a budget of resources over a very large number of items. The interface has two related innovations. First, it asks participants to indicate how they allocate the resources by drawing a curve over a stylized representation of the many items. Second, because the curve represents relative, rather than absolute, levels of resources, it remains visually stable when a participant revises any specific allocation, while the program automatically enforces the budget constraint. We describe the application of the interface to a Blotto game with more than 200 items, but the approach can be adapted flexibly to general problems of resource allocations over many items, under a budget constraint.

1 Introduction

Economics studies the allocation of scarce resources among competing claims. In many applications, the focus is on the distribution of a budget over multiple items–a basket of goods purchased on a market, in individual choice problems; a multi-unit auction or multiple simultaneous contests, in strategic situations.

In lab experiments, to avoid confusing participants with too complex a task, the number of items under consideration is typically limited to a small number. For example, in their 2015 survey of experiments on different models of contests, Decheneaux et al. cite a total of 201 lab studies. Most study a single contest, but even when the focus is on multiple simultaneous contests, the number of contests is typically three or four, only rarely six or eight.[1] Some influential

*We thank Gian Paolo Vernocchi and Simone Daminato at Destinybit for their programming expertise and creativity, Jean Francois Laslier for inspiring conversations on the initial design, and the Institute for Social and Economic Research and Policy (ISERP) at Columbia University for financial support.

†Columbia University, NBER and CEPR, ac186@columbia.edu

‡Paris School of Economics and Ecole des Ponts Paris Tech, olivier.compte@gmail.com

[1]The larger numbers refer primarily to experiments on Blotto games. See for example, Avrahami and Kareev (2009), Arad and Rubinstein (2012), Chowdhury et al. (2012), Casella et al. (2018).

applications of these models, however, concern problems with large numbers of contested items. A classic example is Myerson (1993): candidates compete to win an election; each has a budget to distribute freely over a continuum of voters, and each voter votes for the candidate who offers the most generous transfer. Myerson's article is a fundamental contribution to voting theory, arguing that politically induced inequality among ex ante identical voters may be the natural result of democratic elections. It is also a very difficult theory to test in the field. And yet, it has never been tested in a lab, and neither have equally influential follow-up works that share the focus on politicians competing over voters or electoral districts (Dekel et al. (2008), Lizzeri and Persico (2001) and (2005), Sahuguet and Persico (2006), Snyder (1989)). The difficulty is in designing an experiment that asks participants to allocate resources over a very large number of items, and still keeps the task transparent enough for them to think rationally over the possible choices.

The purpose of this note is to propose an experimental interface that addresses this challenge. The interface is driven by two main ideas. First, in contrast to asking participants for their allocations item by item, as is typically done, participants are asked to draw a curve over a stylized representation of the many items. The curve can be modified freely, with the budget constraint automatically enforced by the computer. Second, the curve depicts *relative* levels of resources over the multiple items, rather than absolute levels. When a participant adjusts the curve, for example by increasing resources on one of the items, the budget is automatically rebalanced so as to satisfy the constraint while maintaining all other relative values constant. As a result, the curve remains visually stable. If the curve showed absolute levels, satisfying the budget constraint would require the whole curve to shift in response to any change, creating a very difficult environment for the subjects.

To our knowledge, the existing experimental interface closest to our goals is the Distribution Builder (Sharpe et al. 2000, Goldstein et al. 2008), an interactive tool used to elicit respondents' preferences over investment portfolios.[2] The Distribution Builder asks subjects to provide a desired distribution of probabilities over future wealth under a cost constraint. The interface computes the least costly portfolio that generates the desired distribution given the prices of available assets, and each subject modifies (by "tatonnement") the desired distribution to meet the cost constraint.

Besides the graphical appearance, our interface and the Distribution Builder have other differences. First, our curve is not a probability distribution; it is a direct allocation of resources. Second, we ask subjects to draw a continuous curve indicating relative levels of expenditure across item values. Thus we call attention to the curve's shape and slope, as opposed to the vector of absolute expenditures, item by item. Third, we implement the budget constraint very differently. In our case, subjects are allowed to draw any curve at all, and the budget constraint is imposed ex post by the computer, when translating relative allocations into absolute levels of expenditure. The Distribution Builder, on the

[2]We thank an anonymous referee for bringing the Distribution Builder to our attention.

other hand, prevents subjects from finalizing the distribution unless the budget constraint is met. Because asset prices are an important component of the exercise and are not described to the subjects, experimenting with the budget constraints allows subjects to learn them. In our case, there are no hidden prices, and the budget is a straightforward sum of allocated resources.

Because we are not familiar with other experimental interfaces asking participants to express their allocations by drawing a curve and because the range of problems to which the approach could be applied is large, we hope that a detailed description of the interface may be useful to others. The platform was constructed by Destinybit (https://destinybit.com/), a game design company. We describe in the Supplementary Material how to download the program and how to parametrize it for the specific game we studied. However, the logic behind the platform's design is more flexible than the particular application and can be reproduced in different programs. It is such logic, more than the software itself, that is the purpose of this note.

2 The experimental interface

The interface was built for the experiment discussed in Casella et al. (2024). For the sake of concreteness, we briefly describe the game here. We study a Blotto game where two opponents are each endowed with a budget of resources R_i, to be distributed over a continuum of battlefields. The battlefields have different values $v \in \mathcal{V}$, drawn independently from a common distribution $f(v)$. Independence holds both across battlefields, for given player, and across the two opponents, for given battlefield. Each battlefield is won by the player who allocates to it more resources. A player's strategy is a resource allocation function $b_i(v)$ that maps the battlefields' values into resources allocated to each value, under the constraint that the resource constraint be satisfied, or $\int_{\mathcal{V}} b_i(v)f(v)dv = R_i$. The object of interest is the allocation function $b_i(v)$. The theory delivers specific predictions on the shape of such function in equilibrium; the challenge was eliciting such a function experimentally.[3]

In the experiment, we framed the battlefields as assets to be defended and represented them as marbles of different values. All marbles a player owns eventually clash with another player's marbles in one-to-one duels, but before duels occur, players make their marbles "stronger" by endowing them with defenses from a known budget of defenses. In each duel, the marble with lower defenses is destroyed; the stronger one survives, and its owner earns the marble's value as points. The platform was constructed by Destinybit (https://destinybit.com/), a game design company. As mentioned, the Appendix provides a link for downloading the program, as well as instructions on how to make some modifications to the original game, for example by changing the distribution $f(v)$, or allowing for asymmetric budgets $R(i)$, or introducing correlation in values.

[3]In Blotto games, the total amount of resources to be distributed is fixed. More generally, in problems with a budget constraint, the resources' residual use would be depicted on the platform as an additional item.

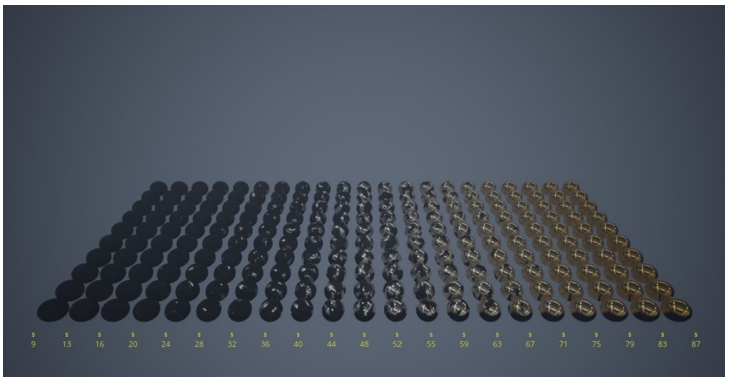


Figure 1: Experimental interface: opening screen. $f(v)$ uniform; 210 marbles.

At the start of the experiment, each participant sees the marbles with which each is endowed. These are the assets, of different values, the participant will have to defend. With $f(v)$ Uniform, the first screen corresponds to Figure 1.

To approximate the continuous distribution, players must be endowed with as many marbles as possible. But when the number is large, handling each marble separately (appreciating its value and choosing the desired level of defenses) is infeasible. Marbles must then be organized in classes. In practice, players are faced with 21 classes of marbles, organized in columns, with each column corresponding to a different value, and all marbles in the same column having the same value. Thus the marbles can take any of 21 different values, roughly suggested by their color, from black to gold, and reported more precisely by the number at the bottom of each column. In the case of a Uniform distribution, as in the example of Figure 1, all columns have the same number of marbles (here, 10), and thus there is a total of 210 marbles. All marbles will fight duels. [4]

A player's task is the allocation of defenses to the player's marbles. Defenses are not represented through objects but graphically, through a curve each player draws, the literal experimental counterpart of the $b(v)$ bidding function of the theoretical model. We asked each participant to allocate defenses by moving the cursor of the computer mouse smoothly over the image of the marbles. All marbles in the same column receive the same amount of defenses, represented by the height of the curve over that column. Figure 2 is an example we used in the instructions for the case of a triangular distribution $f(v)$.

Although only 21 different values are thus relevant when allocating defenses, the game is much more intuitive if presented as if the value distribution were continuous. When asked to draw a curve, players focus on the shape of the curve, that is, on the slope of the allocation function–the change in defenses as the values of the marbles increase–as opposed to choosing the exact numerical

[4]Because each marble is matched in duel against a random opponent, the multiplicity of marbles of the same value works to ensure that the points earned by the player from a class of value v approximate ex ante expected earnings from a random match for a marble of value v.

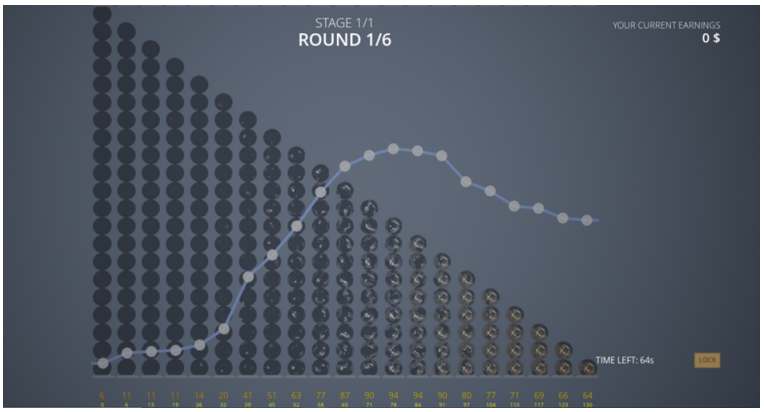


Figure 2: Experimental interface: distribution of defenses. Example with $f(v)$ triangular; 231 marbles.

amount of defenses assigned to each of the 21 values. This short video gives an idea of the interface. We describe the different steps in the next section.

3 Eliciting the defense allocation function

Asking participants to draw a curve creates two related challenges. First, the visualization of different allocations needs to be somewhat robust to the scale of the graph: the image on the screen must remain legible whether the player devotes all defenses to a single column, or divides them less asymmetrically. We set $R_i = 10,000$ to allow for a fine enough variation in assigned defenses while still expressing them in integer numbers. As a result, however, the absolute value of defenses assigned to each marble could vary widely, creating difficulties with the visualization. In the Uniform case, for example, with $R_i = 10,000$, the amount of defenses assigned to a marble could vary between 0 and 1,000 (taking into account that all marbles in the same column must receive the same defenses). But then the difference between, say, 25 and 50, would be impossible to see, even though one number is twice the other. Second, the defense budget must be satisfied. If we asked participants to draw absolute resource allocations, then revising the defenses assigned to one column would force other allocations to change as well, to enforce the budget constraint. The global change in the curve would be sure to generate confusion. Asking participants to report graphically the *relative* allocation of defenses per column addresses both problems.[5]

Call $Rmax_i$ the maximal defense assigned by player i to any marble (note: any marble, not any column, since, depending on the distribution $f(v)$, columns can have different numbers of marbles, as they do in Figure 2). $Rmax_i$ provides

[5] The difficulty of seeing small differences in allocations is reduced by an order of magnitude. It remains when allocations are very small and quite similar, but also economically not very meaningful.

the anchor against which relative allocations are calculated. Its value, in absolute terms, is calculated by the computer for any allocation of reserves, so as to satisfy the budget constraint, and adjusts with any change to the allocation.

The vertical axis corresponds to fractions of $Rmax_i$ and ranges between 0 and $100\%Rmax_i$. In Figure 2, the curve is three times higher in column 7 relative to columns 2, 3 and 4, and thus the curve assigns three times as many defenses to any marble in column 7, relative to any marble in columns 2, 3, and 4. The larger number at the foot of each column indicates the absolute defenses assigned to each marble in the column, while the smaller number reminds the player of the value of the marbles in the column.

At the start, before actively allocating defenses, the subjects see a horizontal line over all columns, an initial allocation curve assigning equal defenses to all marbles.[6] During play, participants modify the curve. Participants modify a point on the curve by clicking on it and raising or lowering it.

The curve is anchored by points we call *anchors*. A point on the curve becomes an anchor by being clicked on, giving the visual impression of "grabbing" the curve at that point. Only points corresponding to a column can be anchors, but because the columns fill compactly the whole screen, the player's perception is that any point on the curve can be clicked on and modified. In addition, although each individual point can only be moved vertically, the density of columns on the screen means that when drawing a continuous curve the experience approximates closely the sense of drawing without constraints.[7] In practice, as a participant draws a curve by moving the cursor of the mouse freely over the marbles, the points corresponding to all columns typically become anchors.

Anchors matter because when participants raise or lower one point, the curve only changes between the two closest anchors, on the point's either side: the segments of the curve between the point and its closest anchors on the point's two sides adjust by linear interpolation. The other segments, outside the two closest anchors, do not move. When, as most likely, points corresponding to all columns have become anchors, modifying the curve over one column has no effect on the height of the curve over any other column. To reiterate our main point, this is possible because the curve represents relative defenses. If it represented absolute defenses, the entire curve would move in response to any revision because all allocations of absolute defenses (including $Rmax_i$) would have to change to satisfy the budget constraint.

To calculate the amount of absolute defenses assigned to any column, the program solves two problems. First, the program determines the defenses assigned to each marble, expressed in percentage of $Rmax_i$, by linear interpolation between any two anchors. Second, the program calculates $Rmax_i$.

The first step corresponds to the visual changes of the curve, as the player makes any adjustments. For example, if the vertical height of the allocation

[6] In this case, and in this case only, the height of the curve is irrelevant, and participants are warned of this and explained why. All marbles have defenses equal to $100\%Rmax$.

[7] Under the limitation that the curve can only appear on each column once: each column can have only one allocation of defenses.

curve is set at 20 at position (value) 75, and at 100 at position 100, if both are anchors, and there is no other anchor between them, then at position 80 the allocation curve is set automatically at $20 + [(100 - 20)/(100 - 75)] \times 5 = 36$. If we call x the horizontal position (recall that x is the value of marbles at the corresponding column) and y the height, then:

$$y_m = y_l + \left(\frac{y_r - y_l}{x_r - x_l}\right)(x_m - x_l)$$

where m denotes the point of interest, intermediate relative to the anchors to the left (l) and to the right (r). Here $x_l = 75$, $x_r = 100$, $y_l = 20$, $y_r = 100$, and $x_m = 80$. Hence $y_m = 36$.

If the subject decides to move the curve at position 80, then the height at position 80 is determined by the subject, while the heights strictly between position 75 and position 100 (the two closest anchors on the two sides) are determined by the equation above. If $y_{80} = 50$, for example, then:

$$y = 20 + \left(\frac{50 - 20}{80 - 75}\right)(x - 75) \text{for all points between positions 75 and 80}$$
$$y = 50 + \left(\frac{100 - 50}{100 - 80}\right)(x - 80) \text{for all points between positions 80 and 100.}$$

The curve does not change at or below position 75. This is the program's first step.

As a second step, given the defenses assigned to each marble as proportion of $Rmax_i$, the program calculates $Rmax_i$ using the constraint that all defenses must equal the total budget. Given the value of $Rmax_i$, the program then translates all relative defenses into absolute values, reported numerically at the bottom of each column.

If we call n_c the number of marbles in column c, C the total number of columns (21 in our implementations), and y_c the height of the curve at column c, then $Rmax_i$ must solve:

$$\sum_{c=1}^{C} n_c \frac{y_c}{100} Rmax_i = R_i$$

where R_i is i's total available defense budget.

In the example of Figure 2, the total number of marbles is 231, divided in 21 columns with $\{21, 20, ..., 1\}$ marbles each. At the opening screen, $y_c = 100$ for all columns; if $R_i = 10,000$, we have $231(100/100)Rmax_i = 10,000$, or, rounding up, $Rmax_i = 43$. Since at the opening screen $y_c = 100$ (or 100% $Rmax_i$), every marble is allocated absolute defenses equal to 43. Suppose instead, for example, $y_c = 20$ for the first 10 columns, and 100 for columns 11-21. Then:

$\left(\sum_{z=12}^{21} z(20/100) + \sum_{z=1}^{11} z(100/100)\right) Rmax_i = 10,000$, or, after rounding, $Rmax_i = 101$. Each marble in the first 10 columns is allocated defenses just above 20, and each marble in columns 11-21 defenses equal to 101.

As the player manipulates the curve, all larger numbers at the foot of each column, representing the absolute amount of defenses assigned to each marble in the column, necessarily change (to respect the budget constraint), while the curve remains stable beyond the two anchors closest to the point the player has modified. The program stores, for each player, the vector of 21 x, y values along the final allocation curve, as well as the history of revisions made by the player.

The computer interface was designed to maximize the sense of drawing a continuous curve and little practice is sufficient to generate the allocation curve by moving the mouse smoothly over the profile of all marbles. The curve can be manipulated easily–for example, players can delete a point by right-clicking on it (and the height of the allocation curve at that point is then interpolated linearly from the two nearest anchors)[8]–and at any modification the constraint that the entire defense budget be allocated is imposed automatically by the computer. The design allows players to draw a defense allocation function that is intuitive, uses the full budget, and remains stable as details are fine-tuned.

4 References

Arad, A. and A. Rubinstein, 2012, "Multi-dimensional iterative reasoning in action: The case of the Colonel Blotto game", *Journal of Economic Behavior and Organization*, 84, 571-585.

Avrahami, J. and Y. Kareev, 2009, "Do the Weak Stand a Chance? Distribution of Resources in a Competitive Environment". *Cognitive Science*, 33, 940-950.

Casella, A., J. F. Laslier and A. Mace', 2017, "Democracy for Polarized Committtees: The Tale of Blotto's Lieutenants", *Games and Economic Behavior*, 106, 239-259.

Casella, A., O. Compte, and S. Si, 2024, "Dealing with Complex Games: Methodological Notes and an Experiment", presented at the 6th Columbia Conference in Economic Theory, September 12-13.

Chowdhury, S., D. Kovenock, and R. Sheremeta, 2013, "An experimental investigation of Colonel Blotto games", *Economic Theory*, 52, 833-861.

Dechenaux, E., D. Kovenock, and R. Sheremeta, 2015, "A survey of experimental research on contests, all-pay auctions and tournaments", *Experimental Economics*, 18, 609-669.

Dekel, E., M. Jackson, and A. Wolinsky, 2008, "Vote Buying: General Elections, *Journal of Political Economy*, 116, 351-380.

Goldstein, D., E. Johnson, and W. Sharpe, 2008, "Choosing Outcomes versus Choosing Products: Consumer-Focused Retirement Investment Advice",

[8]See the example at time 0:28 in the video, where the curve is dragged downward in the sixth column from the right, but snaps back upward when that point has been right-clicked and has stopped functioning as an anchor.

Journal of Consumer Research, 35, pp. 440-456.

Lizzeri, A. and N. Persico, 2001, "The Provision of Public Goods under Alternative Electoral Incentives", *American Economic Review*, 91, 225-239.

Lizzeri, A. and N. Persico, 2005, "A Drawback Of Electoral Competition" *Journal of the European Economic Association*, 3, 1318-1348.

Myerson, R., 1993, "Incentives to Cultivate Favored Minorities Under Alternative Electoral Systems", *American Political Science Review*, 87, 856-869.

Sahuguet, N. and N. Persico, 2006, "Campaign spending regulation in a model of redistributive politics", *Economic Theory*, 28, 95-124.

Sharpe, W., D. Goldstein, and P. Blythe, 2000, "The Distribution Builder: A Tool for Inferring Investor Preferences", Working paper at `https://web.stanford.edu/~wfsharpe/art/qpaper/qpaper.html`, Stanford CA.

Snyder, J., 1989, "Election Goals and the Allocation of Campaign Resources", *Econometrica*, 57, 637-660.

The Beads Program by Destinybit

A. Preamble: Duels, Teams and Payoffs

The mechanics of duels and payoffs were designed with two concerns in mind: minimize the impact of randomness and support subjects' learning, attention, and incentives in treatments with asymmetric resources. Subjects (even in number) are divided into two teams. All members of the same team play against the same opponent, from the opposite team. Payoffs are determined by relative performance within a team. The chosen opponent, called the opposite team "champion" is the subject with the highest point score in the preceding round (in the first round, champions are chosen randomly).

Each duel is a confrontation between two beads. If there are N beads for each player, the program numbers the N beads $1, .., N$ in order of their values but with progressive numbers also assigned to beads of equal value in the same column. The computer chooses two random permutations of this list and generates a random list of these N numbers for Team 1, and one for Team 2. (All permutations have equal probability). Call them List1 and List2. The beads of each player in Team 1 are sorted in the order of List1, and each duel the player faces is against the beads from the Team 2 champion, sorted according to List2. Similarly, the beads of each player in team 2 are sorted in the order of List2, and each duel the player faces is against the beads from the Team 1 champion, ordered according to List1. For example: suppose List1 = 32, 1, 88, 55, 23,..; List2 = 2, 16, 85, 72, 12,... Then, for each player in Team 1 bead 32 faces bead 2 from the champion of Team2; bead 1 faces bead 16, etc.. Similarly, for each player in Team 2, bead 2 faces bead 32 from the champion of Team1, etc..

Players earn points by having their beads survive duels. Earnings by rounds are calculated comparing own round points to the points earned by all members of the own team. Each subject is paid a constant Base Payoff g plus a fraction of a prize, proportional to the subject's position relative to the max and min in the number of points within the subjects' own team, calculated for each round and then summed over all rounds. In each round and each team, define the max number of points as $\bar{p}^r$ and min as $\underline{p}^r$. For each subject i, define the distance index d_i^r measured as $p_i^r = (1 - d_i^r)\underline{p}^r + d_i^r \bar{p}^r$. Total payment (up to show-up fee) is $\sum_r d_i^r \bar{G} + g$ where $\bar{G}$ is the highest prize. Both g and $\bar{G}$ are set by the experimenter.

B. How to Load, Customize, and Start the Program

1. Download and extract the files from:

   https://www.dropbox.com/s/b5exwbm86e3e09j/BeadsDefence-31Aug21.zip?dl=0

2. This creates a WindowsNoEditor folder. Open it and double click on the Beans application file.

3. This opens a game window. On the experimenter's computer, click on Change Mode and switch to Operator Mode.

4. Enter the password: SaltAndPepper (case sensitive). The experimenter can now parametrize the session.

5. Set the value of G upperbar and of Base Payoff (g), calibrated to desired monetary payoffs.

6. Click on Add Match. The program loads the file test.gcf and lets you customize the game - you can change the number of players (which must be even); the number of rounds; the distributions of beads; the budget of defenses (equal or different for players in the two teams); the relative values of the least and most valuable beads; the correlation between the values of each pair of beads engaged in duels. A few notes:

    a. The program defines Match as a specific parametrization. You can include multiple Matches in the same experimental session. The program will pause after all rounds of the first Match are completed and ask you to initiate the second Match (thus allowing time for a different set of instructions). During play, the Operator's window will show the number of the current Match (called Stage Count), as well as the cumulative number of rounds played (Rounds Count).

    b. The number of players corresponds to the number of client computers the experimenter chooses to initiate (see below).

    c. The number of rounds is set manually in the test.gcf file.

    d. The distribution of beads values is also set manually. Nine distributions are available and are described below.

    e. The range of beads values is flexible and is set via LeftGold Distrib and RightGold Distrib. LeftGold Distrib and RightGoldDistrib must be numbers between 0 and 1 and fix the relative values of the most/least valuable beads. The absolute values of the beads are then calculated by the program and depend on the distribution chosen (so as to satisfy the pre-assigned fixed total value of all beads). We recommend leaving Right at 1 and setting Left freely (we used either 0.1 or 0.5).

    f. The game is clearer with more defenses. We set defenses at either (10,000; 10,000) (for symmetric games), or at (10,000; 8,000) (for asymmetric games).

    g. Correlation specifies whether beads are ordered by value before the duels. Setting Correlation equals to 0 is equivalent to imposing independence: beads are matched in duels randomly, independently of values. Setting Correlation equal to the total number of beads is equivalent to full correlation: each bead is matched in duel with an opposite bead of equal value. The program can also implement partial correlation.

The program starts by creating the two permutations as described in the Preamble. Each permutation is then divided into sublists of m beads (where m is the correlation parameter), and ordered in increasing value within each sublist. For example, if $m = 5$, suppose the first sublist for team 1 is 92, 3, 51, 16, 28, and the first sublist for team 2 is 51, 53, 8, 94, 6, then the ordered sublists become 3, 16, 28, 51, 92 and 6, 8, 51, 53, 94. Hence for each player of team 1, bead 3 is matched with bead 6 of the champion of Team2, bead 16 with bead 8 etc.. Note that $m = 0$ and $m = 1$ are equivalent (both induce independence), and the lower is m the lower is the correlation in the values of the duelling beads.

h. The Match (or Matches) parametrization is recorded in the file test.gcf. If desired, the file can be saved with a distinguishing name (test_XXX.gcf) and loaded directly in future sessions.

7. After parametrization is completed, click on Open Lobby.

8. Go back to the Beans application icon. Double click and create a client.

9. The IP number is the IP number of the Operator computer. (To test the program on one's own computer, insert the internal IP: 127.0.0.1).

10. Go back to the Beans application icon to create each client. The program needs an even number of clients.

11. Any subject on a client screen can click on Instructions. The instructions give a synthetic version of the rules of the game. They show a specific distribution of beads but specify that what the subject will see in the experiment may be different. This can be emphasized orally, if the Match has a different distribution. Alternatively, the Operator may give instructions orally and underplay the ones included in the program.

12. When ready, each subject clicks on Ready.

13. The Operator can then click on Start Game.

14. To move from one Match to the next, the Operator needs to advance the game on the Operator's screen.

C. Description of the distributions

The program lets you choose among nine different distributions of beads' values, depicted in Figure A.1 at the end of the Supplementary Material. Recall that all beads in a given column always have the same value and that the support of absolute values is determined by your choice of Left/Right Gold Distrib and the distribution. The options are:

1. Rectangle21x10: 21 columns, 10 beads per column, 210 beads in total. Rectangles over the full support correspond to Uniform distributions. (Image 1)

2. Triangle21x40: 21 columns, max height 40 beads, 230 beads in total. Beads per column are: {1,1,1,1,2,3,8,18,24,36,40,36,24,18,8,3,2,1,1,1,1}. Steep symmetric triangles correspond to symmetric Beta's with high concentration. (Image 5).

3. HalfRectangle21x20: 21 columns, the first and last 6 columns have 1 bead; the center 9 columns have 20 beads each. 192 beads in total. The distribution mimics the high central mass of the triangle distribution but is easier to visualize and probably less confusing to the subjects. (Image 2)

4. Rectangle15x10: 15 columns, 10 beads per column, 150 beads in total. (Image 3)

5. Rectangle11x10: 11 columns, 10 beads per column, 110 beads in total. (Image 4)

6. Triangle15x40: 15 columns, max height 40 beads, 182 beads in total. Beads per column are: {1,1,1,2,8,22,36,40,36,22,8,2,1,1,1}. (Image 6)

7. Triangle11x35: 11 columns, max height 35 beads, 101 beads in total. Beads per column are: {1,1,2,5,24,35,24,5,2,1,1}. (Image 7)

8. ATB21x21 (Asymmetric Triangle Bottom): 21 columns, max height 21 beads, with the number of beads declining by 1 progressively, from the lowest value to the highest value column; 231 beads in total. Beads per column are: {21,20,19,...,1}. (Image 8)

9. ATT21x21 (Asymmetric Triangle Top): 21 columns, max height 21 beads, with the number of beads increasing by 1 progressively, from the lowest value to the highest value column; 231 beads in total. Beads per column are: {1,2,3,..., 21}. (Image 9)

D. Description of the output file

The output is a csv file that records, in addition to the set parameters for each round, the curve drawn by each player, both in relative and absolute values, the outcome of each duel, and the subject's round and cumulative payoffs. The file also reports, for each subject, the number of seconds before finalizing the curve, the time, number and details of all revisions to the curve, and all information about duels. The annotated copy of an output file in Figure A.2 below. The format is:

- number of players; number of rounds.

For each round:

- Round id; Distribution shape id; Distribution shape name; Defences team 0; Defences team 1; Seconds available for curve; Correlation; Number of beads; Number of columns; LeftGoldDistr value; RightGoldDistr value.
- Champions id Team 0; Champion id team 1.
- Order of beads for duels Team 0.
- Order of beads for duels Team 1. (Note that the order of beads in duels is kept constant across players in each team to reduce differences due to randomness).

And then for each player:

- Player id; Team id; Seconds used for curve; Total points; Round points; Round payoff distance; Cumulative payoff distance; Total beads won; Number of curve revisions.
- Relative defense allocations.
- Absolute defense allocations.
- Number of beads won per column.
- History of curve revisions. One line per revision with format: Time; Relative allocations per column.
- Duels data, one line per bead, with format: Subject id; Bead value; Bead defence; Opponent id; Opponent bead value; Opponent bead defence; Outcome.

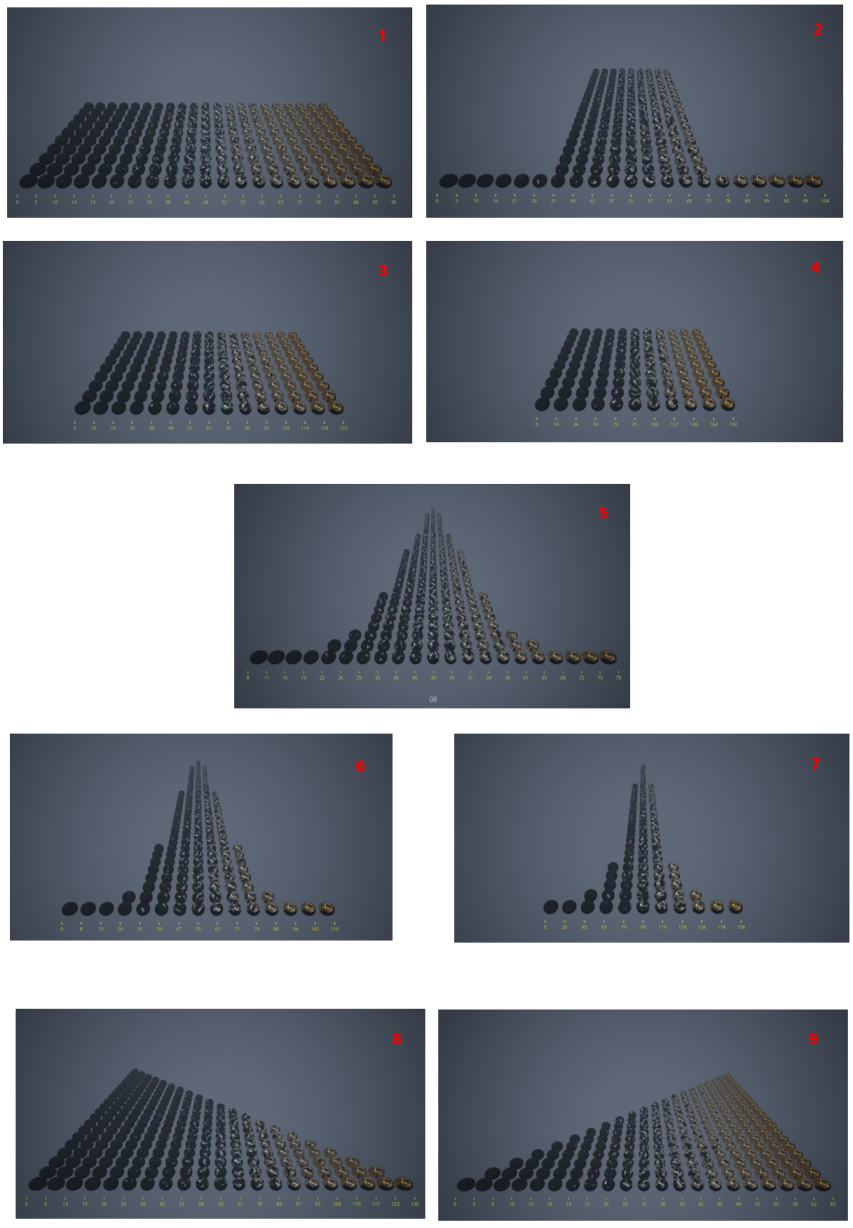
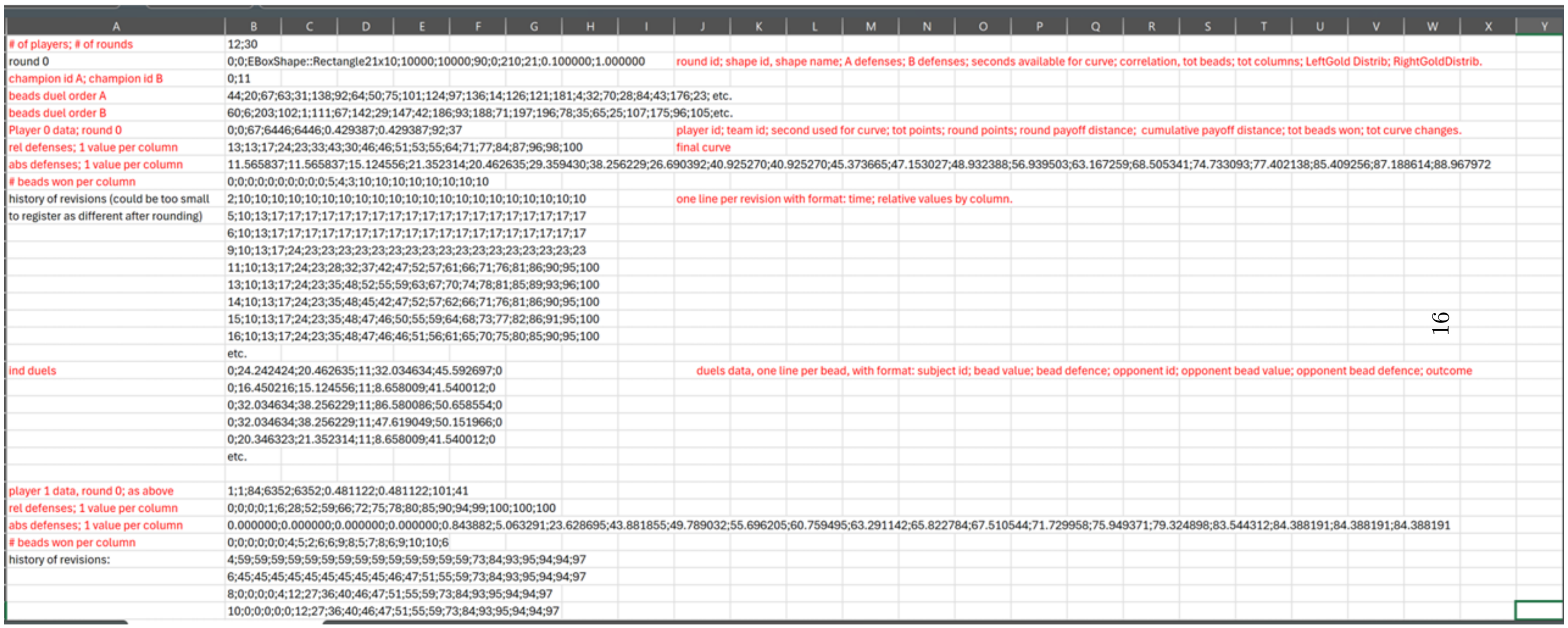


Figure A.1. Preprogrammed Distributions



A	B	J
# of players; # of rounds	12;30	
round 0	0;0;EBoxShape::Rectangle21x10;10000;10000;90;0;210;21;0.100000;1.000000	round id; shape id, shape name; A defenses; B defenses; seconds available for curve; correlation, tot beads; tot columns; LeftGold Distrib; RightGoldDistrib.
champion id A; champion id B	0;11	
beads duel order A	44;20;67;63;31;138;92;64;50;75;101;124;97;136;14;126;121;181;4;32;70;28;84;43;176;23; etc.	
beads duel order B	60;6;203;102;1;111;67;142;29;147;42;186;93;188;71;197;196;78;35;65;25;107;175;96;105;etc.	
Player 0 data; round 0	0;0;67;6446;6446;0.429387;0.429387;92;37	player id; team id; second used for curve; tot points; round points; round payoff distance; cumulative payoff distance; tot beads won; tot curve changes.
rel defenses; 1 value per column	13;13;17;24;23;33;43;30;46;46;51;53;55;64;71;77;84;87;96;98;100	final curve
abs defenses; 1 value per column	11.565837;11.565837;15.124556;21.352314;20.462635;29.359430;38.256229;26.690392;40.925270;40.925270;45.373665;47.153027;48.932388;56.939503;63.167259;68.505341;74.733093;77.402138;85.409256;87.188614;88.967972	
# beads won per column	0;0;0;0;0;0;0;0;0;5;4;3;10;10;10;10;10;10;10;10	
history of revisions (could be too small	2;10;10;10;10;10;10;10;10;10;10;10;10;10;10;10;10;10;10;10;10	one line per revision with format: time; relative values by column.
to register as different after rounding)	5;10;13;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17	
	6;10;13;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17;17	
	9;10;13;17;24;23;23;23;23;23;23;23;23;23;23;23;23;23;23;23;23	
	11;10;13;17;24;23;28;32;37;42;47;52;57;61;66;71;76;81;86;90;95;100	
	13;10;13;17;24;23;35;48;52;55;59;63;67;70;74;78;81;85;89;93;96;100	
	14;10;13;17;24;23;35;48;45;42;47;52;57;62;66;71;76;81;86;90;95;100	
	15;10;13;17;24;23;35;48;47;46;50;55;59;64;68;73;77;82;86;91;95;100	
	16;10;13;17;24;23;35;48;47;46;46;51;56;61;65;70;75;80;85;90;95;100	
	etc.	
ind duels	0;24.242424;20.462635;11;32.034634;45.592697;0	duels data, one line per bead, with format: subject id; bead value; bead defence; opponent id; opponent bead value; opponent bead defence; outcome
	0;16.450216;15.124556;11;8.658009;41.540012;0	
	0;32.034634;38.256229;11;86.580086;50.658554;0	
	0;32.034634;38.256229;11;47.619049;50.151966;0	
	0;20.346323;21.352314;11;8.658009;41.540012;0	
	etc.	
player 1 data, round 0; as above	1;1;84;6352;6352;0.481122;0.481122;101;41	
rel defenses; 1 value per column	0;0;0;0;1;6;28;52;59;66;72;75;78;80;85;90;94;99;100;100;100	
abs defenses; 1 value per column	0.000000;0.000000;0.000000;0.000000;0.843882;5.063291;23.628695;43.881855;49.789032;55.696205;60.759495;63.291142;65.822784;67.510544;71.729958;75.949371;79.324898;83.544312;84.388191;84.388191;84.388191	
# beads won per column	0;0;0;0;0;0;4;5;2;6;6;9;8;5;7;8;6;9;10;10;6	
history of revisions:	4;59;59;59;59;59;59;59;59;59;59;59;59;59;59;73;84;93;95;94;94;97	
	6;45;45;45;45;45;45;45;45;45;46;47;51;55;59;73;84;93;95;94;94;97	
	8;0;0;0;0;4;12;27;36;40;46;47;51;55;59;73;84;93;95;94;94;97	
	10;0;0;0;0;0;12;27;36;40;46;47;51;55;59;73;84;93;95;94;94;97	

Figure A.2. Output File Annotated