Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Make millions using this old poker trick

Tim Weber, Jan Speckien, Patrice Gobat, Lionel Gulich

Zurich
December 2016

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Tim Weber

Jan Speckien

Patrice Gobat

Lionel Gulich

Contents

1 Abstract

In the recent years Texas Hold'em has been increasingly popular amongst all the age groups. Therefore our aim in this study is to research how different strategies perform against each other in a heads up. In a next step we invastigated the change in outcome if one player undergoes a learning process.
Hence we wrote a program using MATLAB that simulates a game of Texas Hold'em in which all feasible strategies compete against each other. In our model the strategy is represented by a single variable called risk factor that holds the willingness of taking risks. The scale of the variable reaches from zero -extremely active- to one -extremely passive-. For the latter part one player can adjust his risk variable based on different learning algorithms, which then are compared with each other.
The results show that up to a risk factor around 0.4 and 0.55 the more passive player prevails. Then there is a short transition period in which both players are evenly matched. After that we can see a reversed outcome where the more active clearly dominates. Concering the learnings algorithms we observed a significant increase in the average number of games won. This indicates that with an elaborated strategy also in real Texas Hold'em an important increase in the average games won can be achieved.

2 Individual contributions

The commonly used models for Texas Hold'em are mostly based on stochastical probability, which makes the implementation of algorithms highly complicated. Thus we decided to write our own simulation in order to simplify the problem using as few variables as possible. All further results and conclusions are based on our program.

3 Introduction and Motivations

Texas Holdem is a variation of the card game Poker and is throughout casinos, tournaments and private people amongst the most popular of its kind. The goal of the game consists in winning all the money, or from here on called chips (virtual currency), from the other players at the table. As soon as someone has no more chips left, he lost and has to leave the table. Every round consists of different stages of card dealing and then successive betting. In each stage a player can decide if and how much chips he wants to play according to some rules and by evaluating their private cards called hand and the cards open on the table named sequentially flop, turn and river. The total amount of chips from all the players in one round is called pot. Every round ends with either one player getting the whole pot or sometimes

with a split of the pot, if two or more players have an equivalent score in the end of the round. The score is the highest possible combination of the private and public cards available according to a fixed ranking.
The game can be played with a varying number of people and for simplicity be subdivided into two parts: Group stage and Heads up, whereby the main difference lies in the number of people playing. In a heads up the last two remaining players are gambling for the overall victory and therefore every game of Texas Holdem will end with a heads up.
Every player has his own strategy, where some players like to play more conservative by not betting often and waiting for better cards and others play more aggressive by betting more often and trying to bluff his opponent. An important aspect of the game is the blinds, since they force the players to bet a fixed amount before the first cards are dealt. The blinds move forward by one player after every round. They guarantee that no player only can wait for the perfect cards and that there are always chips in the pot that can be won.
We all agree that Texas Holdem is about knowing your opponent and play smarter or better than him. Therefore it is not possible to have a fixed strategy. One has to be able to adjust his strategy depending on the way the opponent is playing and vice-versa.

Hence we all often wondered what would be good strategies to best adjust to ones opponent and in particularly how effective such strategies are. To make a valuable assumption about the effectiveness of such a strategy, one would need to play an enormous amount of games to formulate a statistically speaking valid statement. If a whole game or at least some parts of it can be simulated repeatedly under different circumstances and with varying strategies, then one could gain deeper insights into Texas Holdem and possibly develop an applicable strategy or at least compare different approaches.

4 Model and implementation

This section serves to understand how we implemented the simulation. Expressions are introduced that we use throughout the report. One can find the whole program on GitHub [1] in the folder code.

[1] https://github.com/atlas000/project-poker-msssm

4.1 main.m

The script ultra-main plays all the different risk factors against each other and saves the amount of wins by player1 in a matrix.

The risk factor determines the character of a player. It is stored with a number between 0 and 1, whereas a smaller number represents a riskier player that plays a hand even if his score is not that high and vice versa.
The function main.m serves as an interface where one can decide on the settings through variables which then get passed on to the function game.

4.2 game.m

The function game.m is used to call the functions headsup/2. Like in real poker game.m continues to deal new hands until one player has run out of money. It is alternately calling two functions that differ only in which player has to pay the blind.

As described above, a blind is essential to the game of Texas Hold'em. A player has to pay some money into the pot without having seen his cards yet. That way if a player is very passive, he looses money anyway and has to become more active at some point.

In our model only a single fixed blind has been implemented, since in a heads up only the difference in blinds is relevant.

4.3 headsup.m

The headsup.m function simulates one hand which consists of up to four rounds. With every new card unveiled each player faces the decision whether to place a bet or not. Their action is determined by comparing their risk factors to their score. In total this scenario occurs up to four times per hand. If both players want to play until the end, a showdown determines the winner. Meaning that they have to show their cards and the one with the better score wins the hand and receives the pot.

In the first round, the "preflop", one player has to pay the blind and each player gets a set of playing card.

The main aspect making poker a complicated game to simulate is its use of cards. Hence our main goal was to find a model, which did not require the implementation of a deck of cards. In order to achieve a similar effect every player is given a random number, called ”card value”. The number reaches from 0 to 1 and represents the quality of the hand, whereas 0 would be the worst hand and 1 the best.

In the next three rounds the ”flop”, ”turn” and ”river” new cards get unveiled on the table. So the score of the players’ hands has to be adjusted.

4.4 adujstCardValue.m

At the heart of this simulation lies the adjustCardValue function, which recalculates the card value of a player.

Important criteria are that the cards correlate with each other, so if a player has good cards it is likely that his hand is still good when a new round is played. Also it should be possible for a value to reach the whole spectrum of card quality from 0 to 1. This is important because if one has a bad starting hand it should still be possible, with some luck, to get the best hand possible after the flop. Hence every value should be possible to receive but not with the same probability.

4.5 Learning models

An important part of real Texas Holdem is to obtain an advantage by estimating the strategy of your opponent and to react accordingly. In this model two different approaches for learning algorithms have been chosen to optimise a player’s strategy, under the assumption that the oponent keeps his risk factor constant.

- One approach is minimise the losses and thus maximise the profit of a player. This model analyses a player’s own performance.
- The other approach was to first analyse the play of the opponent and then choose the optimal strategy to best counter the other player. This of course is only possible if all outcomes for all strategies are known.

4.5.1 Iteration model

This learning algorithm analyses the loss of the last hand, because if the last hand has been won there is no need to adapt the strategy. So after every hand one of the following scenarios takes place:

Validation	Interpretation	Action
CapitalP-$(t+1)$ >= Capital(t)	The player had equal or better cards, or the hand has not been played	riskfactor r will not be changed
CapitalP-$(t+1)$ == Calpital$(t)-4$	The player has lost after the showdown and is thus to agressive	r=r+0.012
Else (CapitalP- dropped by 1,2 or 3)	The player is to passive, since it did not come to a showdown	r=r+0.003

4.5.2 Threshold model

The target of this learning model is to determine the risk factor from the oponent, which then enables to find the optimal risk factor for the learning player. Everytime a showdown is conducted the learning player gets to know the score from his oponent . Because the oponent has continued the hand with this score until the showdown, it is known, that the risk factor from the oponent has to be lower than said score. With every showdown executed the possible values for risk factor of the oponent can be narrowed down, leaving a threshold which has to be undershot by the oponent's risk factor.
The optimal risk factor for the learning player can then be determined with the data obtained by previous simulations. This data includes the win probality of both players for all possible combinations of riskfactors between both players. Determining the optimal risk factor then only is a matter of finding the maximum for a given oponent strategy.

4.5.3 Count and gauge model

The main goal behind analysing the opponents way of playing is to correctly estimate his risk factor, in order to adapt ones risk factor afterwards. This algorithm is accomplished in two consecutive steps:
Firstly it keeps track of the number of times the opponent decides to play the initial round when he is not forced to do so by the blind. The ratio between this value and the total amount of rounds played can then be used as a measurement to estimate

the opponents risk factor:

$$\text{risk factor} \approx 1 - \frac{\#\text{ rounds}}{\#\text{ rounds played}}$$

The second part consists of finding the optimal counterpart to the estimated risk factor. For that reason we evaluated results obtained by previous simulations. The data obtained indicates that for every risk factor chosen, there is at least one optimal counterpart. After extracting these values the algorithm adjusts the own risk factor.

5 Robustness of results and influence of sigma

To draw some conclusions from our model we had to ensure that our model is robust enough. We performed thousand games a thousand times and found a standard deviation of 7.7 for the wins.

To the function adjustCardValue.m a sigma for the flop is needed that leads to a big variety of outcomes because three new cards are dealt in that round. Still the difference in the card value before and after new cards cannot be too wide, otherwise all correlation is lost. In the next two rounds, the turn and the river, where only one card is dealt the current card value has to be somewhat stable.

We chose the combination of 0.4/0.2/0.2 to be the one that represents a game of Texas Hold'em the best.

We ran some simulations with different sigmas and saw that for each combination the resulting graph has the same shape but is shifted to the left or the right of the x and y scale.
Sigma have been tested on their reach from 0 to 1 and lowest/highest possible value for each risk factor. The results can be found on GitHub[2].

6 Simulation Results and Discussion

6.1 Results of generic model

6.1.1 Results with blinds

The figure 1 shows how many games player1 has won out of a total of 1000, when both players started out with an equal amount of chips (50) and a fixed blind equal

[2]https://github.com/atlas000/project-poker-msssm

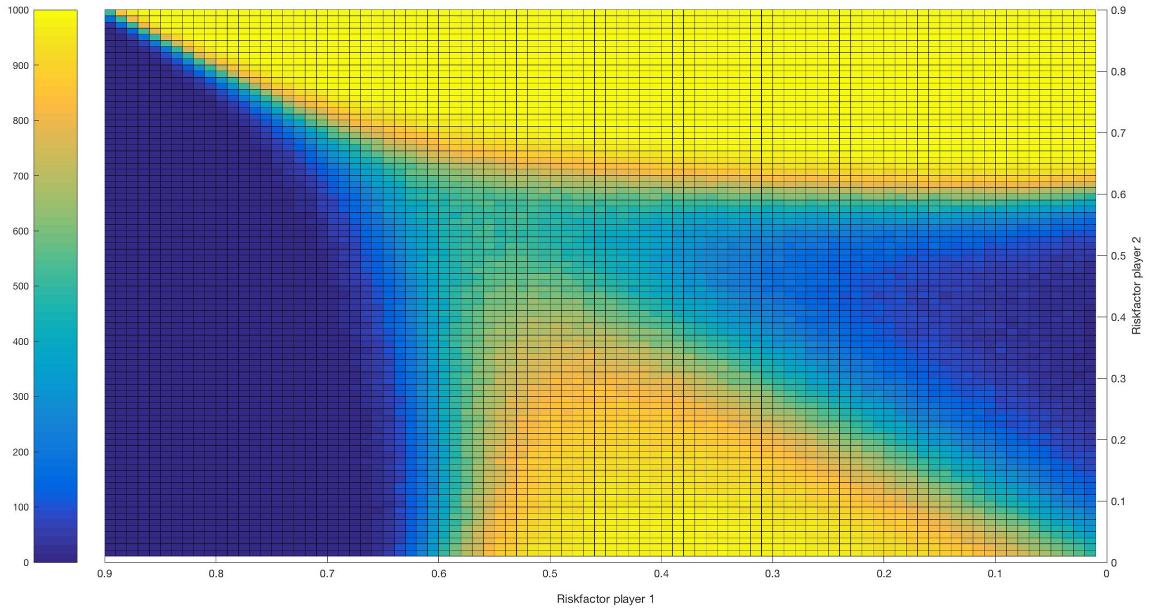

Figure 1: Games won by player1 out of 1000

to the betValue of 1. The colour of every cell ($x = r1|y = r2$) shows the number of games won by Player1 with his risk factor $r1$ against player2 with a risk factor $r2$.
On the diagonal ($r1 = r2$) a green line can be seen, which shows that player1 and player2 have each won half of all games played. Around this line the games won by a player are antisymmetrically distributed, which means that both players did equally well when using the same risk factor (strategy) r.
A very passive strategy ($r > 0.63$) could not win any games, except against a more passive player. In this case, the less passive strategy occurred to be superior.
An aggressive strategy ($r < 0.5$) performed better. It could win against the very passive strategy (r opponent > 0.63). However, it clearly lost against all less aggressive strategies under 0.63 ($r < r$ opponent < 0.63).
Thus, there are two different patterns visible in Figure 1. One pattern ($r < 0.63$), where the more passive player wins. And a second pattern ($r > 0.5$), where the more active player wins.
Between these two patterns lies a transition area, where both risk factors are between 0.5 and 0.63. In this area the games were very balanced and not even an optimal counter-strategy did win more than 60% of all games.
An optimal risk factor against most other strategies was found to be around $r = 0.5$.

6.1.2 Results with blinds — Interpretation

The bad performance of the very passive strategy can be explained with two independent mechanisms. One being the blind, that punishes the more passive player. The other being the card score dropping below the risk factor before the showdown, leading to a loss of all the chips bet during that hand. The aggressive player plays more games with weak hands, which means, that he will lose more often in the case of a showdown. Inside the area, where the game is balanced, all mentioned effects compensate each other. The model weights the effect of a passive player dropping out after already having invested chips to heavy. In a real game of poker a player stays in the game, due to a comparison between his own odds and the pot. This effect is called pot odds.
Otherwise the simulation shows that the aggressive strategy (loose play) is favorable against an extreme passive strategy (tight play). This reflects the reality where blinds give the passive player a disadvantage in the heads up.

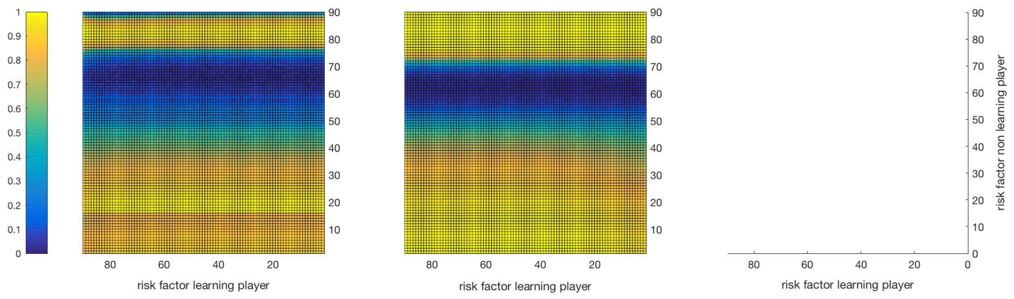

Figure 2: Resulting win matrices for all three learning models. From left to right: threshold, count and gauge, iteration

6.2 Results of learning algorithms

6.2.1 iteration

As seen in figure 3 the algorithm needs around 800 hands in order to leed to a good estimation for the optimal parameter r. This leads to good results for startcapitals above 70 and to very good results at startcapitals above 300 even with a very unfavorable starting strategy like r(0)=0.1.

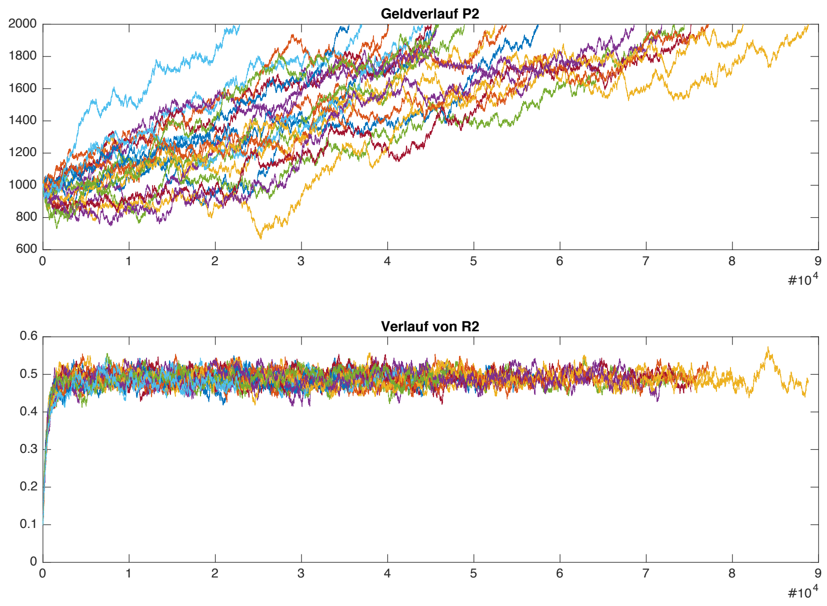

Figure 3: The figure above shows the progression of the capital of the player using the learning algorithm starting at 1000 chips. The figure below shows the progression of the corresponding riskfactor of the same player starting at $r(0) = 0.1$ against a player with a riskfactor of $r = 0.7$.

Advantages

The model has a great potential for different kinds of optimization:

- parameters used in the model can still be optimized in order to perform quicker
- more cases for different scenarios could be made (different action for loss of 1 and loss of two)
- the cards that the player has could be taken into account in order to avoid a change in policy if there has just been bad luck with the cards

Disadvantages

The algorithm is based on a simplified scenario, where the only reason for a loss is besed on the strategy and never on luck. This leads in some cases to a change of policy even when this is not desired. This can be observed in the high standart deviation, even after haveing fond a well estimated r. This behavior leeds to the algorithm loosing against extreemly strong strategies due to its continous slight deviations from the optimum riskvalue.

6.2.2 Count and Gauge model

Advantages

With an increasing number of rounds the estimation continuously gets better

On a long run he will always play the most suitable strategy and optimize his return

It reaches a good estimation of the opponent's risk factor already after a few rounds

The algorithm could also use the rounds where the other player has to pay the blind by counting the number of consecutive rounds he plays and applies the same strategy just by including the models standard deviation into the calculation. This would lead to a faster estimation of the opponent's risk factor

Disadvantages

The algorithm requires a precise knowledge of all possible outcomes and therefore is only applicable in the same circumstances the data was obtained

6.2.3 Threshold model

In figure 2 the results from the learning algorithm can clearly be seen. First of all the graph shows, that the resulting wins are independent from the learning player's risk factor because results are nearly constant along a horizontal line. Next it is visible that the algorithm does not have the same effectiveness for all oponent strategies. For an oponent risk factor between 0 and 0.4 and also around 0.8 the algorithm is really succesfull, as can be seen with the highly yellow areas in the plot, which represent a lot of wins for the learning player. From the green band at an oponent riskfactor of around 0.45 it can be seen that wins are equally balanced around this point. In the blue area for an oponents risk factor between 0.5 and 0.75 wins for the learning player are infrequent. This mean that the algorithm there fails to provide an advantage over the oponent.

By subtracting the resultant matrix from generic model with blinds from that of the learning algorithm it can be visualized where the threshold learning algorithm improves the game results and where it fails to do so. This is shown in figure 4. The highly yellow areas show a drastic improvement in performance, whereas the green areas represent scopes where the performance could only be slightly changed or even remained the same. The deep blue areas show scopes in which the learning players

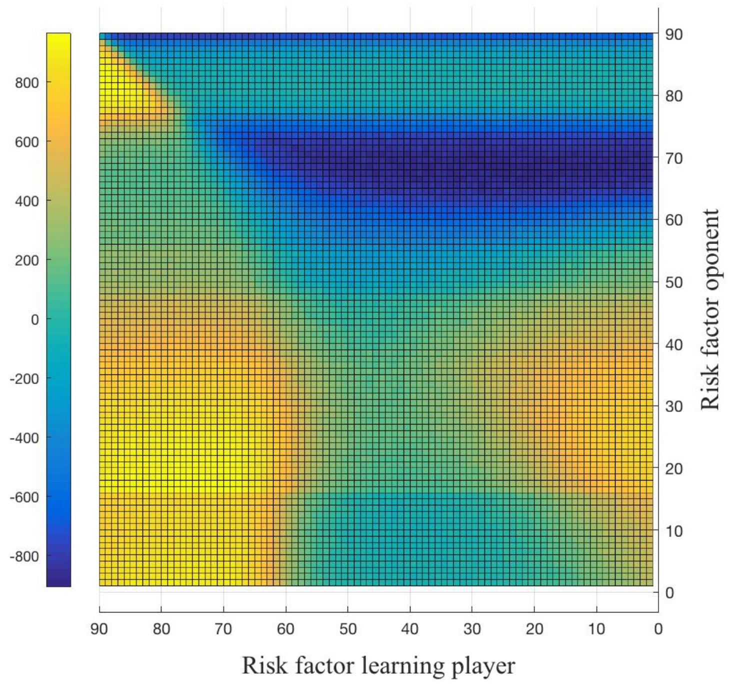

Figure 4: Topview of difference from results of threshold learning model and generic model with blinds

Table 1: Learning models overview figures

	# Games won		# Hands played		∆games won	
	μ	σ	μ	σ	μ	σ
Threshold model	605.7777	301.3260	1565	1854	105.9853	501.7773
Count and gauge model	690.6369	343.1811	709.3543	342.5163	190.8446	477.6301
iteration model	647.55	335.5068	110020	748600	147.7569	198.3177

performance has declined drasticly.

The behaviour of the learning model in the green and yellow areas can be explained as follows: since the performance from the learning player in the green areas was already good with the generic model no big improvement could be realized the difference in wins therefore leveled off in these sections. The yellow areas underly low performance regions in the generic model with blinds, thus a big improvement could realised an the difference in wins rocketed.

What stays to explain are the not expected blue areas. In the blue area at a risk factor of 0.9 the oponent play extremely passive. Hence a showdown seldom occurs and the learning algorithm cannot take effect. However this can only explain the failure of the learning aspect of the algorithm but not the loss of the learning player. The second blue areas behaviour can be explained by the addition of two effects. Firstly the gradient of the generic result matrix is extremy steep, thus already a slight misestimate in the oponents risk factor can lead to a big change in the optimal risk factor for the learning player. Secondly because in the turn the oponent can also play a hand up to 0.15 lower than his risk factor the estimation for his riskfactor can also be shifted downwards by up to 0.15 and therefore lead to a miscalculation in the optimal risk factor for the learning player.

6.3 Comparison of learning algorithm performances

In table 1 some characteristic values for the learning algorithms are displayed. Therefore the mean and standard deviation of the resulting wins matrix, total hands played matrix and the difference in the wins matrix with and without the learning algorithm have been calculated. By comparing the mean of games won it can be seen, that the count and gauge model is the most succesful algorithm in terms of most wins created yet due to the high standard deviation in games won also the most inconsistent. From the standard deviation of difference in games won it can be distinguished, that the iteration is the most balanced overall, leading to small improvements for all different oponent strategies. In comparison therto the threshold and count and gauge model are not succesful against every oponent, but if the are create a high yield.

Lastly the time efficiencys have to be considered: Clearly the count and gauge algorithm is the one leading to a victory the quickest, as its low mean in hands played shows. On the other hand the iteration is the most time consuming. While being the most effective for every oponent its enormous mean amount of hands played leaves it to be essentially useless in practice.

6.4 Robustnes of the Results

When performing 1000 games 1000 times, we found a standard deviation of 7.7 of the wins, Which means the resulting wins after 1000 games are precise on ± 7.7.

6.5 Robustnes of the variables

7 Summary and Outlook

The most important understanding we gained from our simulation was definitely that there is a transition between when it is favourable to play aggressively and when to play rather passively. Under the simulated circumstances the transition happened exactly at a risk factor of 0.49. This can be translated into the applicable technique that if your opponent plays less than every second hand in average, you will on average win more if you play more aggressive than him. On the other side of this barrier, if the opponent plays less than every second hand with playing a little bit more conservative than him the long-term chances are the best. This rule can easily be implemented by just counting the number of times the opponent pays the blind when he is not forced to, divided by the amount of times it was the case.

Our learning algorithm overall increased the performance, but all encountered big problems (? Lio, is it true) in the transfer region. This reflects the complexity of Poker even under simplified conditions and that there is no magical formula even when the opponent plays extremely balanced.

Nevertheless we learned from our approach that by keeping track of the opponents decision, there can be a lot of insights gained about his strategy. After a few rounds played then the easy rule of more or less than 50% of the hands played stated above can be applied to gain an advantage above. In case the outcome of this rule is close to 0.5, the result is to be tread with caution, as the stubborn execution can lead to the exact opposite results as proved by the count and gage algorithm.

This project is developable in many kind of ways. We had thought of some different things we originally wanted to implement but we did not have the time to.
One of them was implementing a blind that would increase over time. This is how it is played in real Texas Holdem with the motive to punish passive play. Because of that it would have been interesting to observe what difference it would have made in our simulation, also since our games lasted up to over a thousand hands and this is far away from reality.

One could also have tweaked the capital/blind ratio in general. Possible outcomes could have been that with a certain ratio the aggressive player gets punished because his loosing hands include bigger pots, but perhaps the conservative player looses more often because games are more fast paced.

Another extremely interesting feature could have been giving the players the option to decide with how much money the wanted to raise the pot. Currently, one is the only option but how would the outcome of a game be if they could bet one, two or three. If that was the case, card evaluation needed to be different. A player now not only has to take his own cards in to the decision if he wants to play or drop out but also how much money his opponent has bet in this phase. Furthermore, would this change have opened the opportunity for a player to bluff his counterpart, this means pretending to have better cards then what he actually has. Now maybe the more active player gets more rewarded for his play style.

The possibility of checking could also have been included. Checking is doing nothing when your opponent has not bet any money. This would open different strategies, because now it is possible that a hand plays out until the end with both players checking all the way to the showdown. This variation would need a distinct playing order and that would put the one, who has to show his cards first in a showdown, into a disadvantage. All these factors could play a role in developing a winning algorithms.

Other variations that certainly would lead to different discoveries but would made the program much more complex are coding ”real” cards with their actual probabilities to show up in a hand but also including more than two players which might would change the power dynamics between active and passive players.

8 References

9 Appendix

9.1 Source Code

9.2 Variable documentation

9.2.1 Variables from generic model

- **Scripts:** ultra_main(um), main(m), game(g), headsUp/headsUp2(h), adjust-CardValue(acv)

- **playerP1, playerP2, general:** Vector with 4 entries representing all important variables from one player.
  1. risk factor for player
  2. capital from player
  3. card value from player, is a random number between 0 and 1
  4. total bet from player
  5. free variable for learning implementation

- **allData um:** Matrix containing number of Wins from player 1 one for varying riskfactors

- **r1, r2 um:** iteration variables representing riskfactors for players 1 and 2

- **betValue m,h:** represents the amount a player can bet on his win or the amount by which the pot can be increased per round per person

- **n m:** iteration variable for determining how many games are being simulated, hence determining the accuracy of the monte carlo approach

- **riskFactorP1, riskFactorP2 m:** variables representing the riskfactors for both players

- **startCapital m:** determines the capital at the beginning of the game for both players.

- **winsP1 m:** amount of total wins by player 1, used as output to the function main.m

- **winner m,g:** stores the the winner of the game simulated: if 0 winner is player 2, if 1 winner is player 1, used as output to the function game.m

- **counter g:** counts amount of hands played in one game

- **decide_who_starts g:** used to determine whose turn it is to start with betting, if 0 player 2 begins, if 1 player 1 begins

- **pot h:** stores the total amount of money betted by both players

- **capP1, capP2 :** output variables to headsUp.m function storing the capital of the corresponding player after having played the hand

- **newRandValue acv:** output to adjustCardValue.m function, stores the newly generated cardvalue

- **sigma acv:** theoretically the standard deviation of a normally distributed random value, here used to determine the range of adjustement for the function adjustCardValue.m

9.2.2 Variables from learning models

Threshold model

- **Scripts:** ultra_main(um), main(m), game(g), headsUp/headsUp2(h), adjustCardValue(acv), adjustRiskFactor(arf)

- **totalCounter m:** used to store total amount of hands played per game

- **playerP1(5) g:** stores the riskfactor of player 2 as it is currently estimated by player 1

- **startRiskFactor h:** input to headsup.m function, stores the riskfactor from player 2 as estimated by player 1 before the respective hand

- **estRiskFactor h:** output from headsup.m function, stores the riskfactor from player 2 as estimated by player 1 after the respective hand

- **newRiskFactor arf:** output from adjustRiskFactor.m function, stores newly generated riskfactor

- **opponentRiskFactor arf:** input to function adjustRiskFactor.m, currently estimated riskfactor from opponent player

- **refSurf arf:** two-variable function which represents amount of wins for player 1 in dependence of the respective riskfactors

- **funVector arf:** parameterisation of refSurf at point of opponentRiskFactor