Advanced Cognitive Modelling
Assignment 1

Matching Penny Strategies

Cognitive Science Masters: Spring 2022

Tobias Christensen | Student reg. no.: 201806880

Melina Vejlø | Student reg. no.: 201806965

Orla Mallon | Student reg. no.: 201807639

Linnea Bendix | Student reg. no.: 201708397

GitHub: https://github.com/Orlz/Advanced_Cognitive_Modelling.git

AARHUS UNIVERSITY

Introduction

The Matching Pennies Game is a popular concept in game theory, where it is used to demonstrate how independent rational decision-makers seek to maximise their payoffs. The game itself is simple. There are two players who are instructed to either match or mismatch their opponents choice, by blindly selecting the '*heads*' or '*tails*' of a coin in a given round. Points are allocated to the players depending on their role and their opponents choice. If the player holds the role of 'matcher', they win the point when both players select the same side of the coin and lose the point when they select opposite sides of the coin. Followingly, the mismatcher loses the point when the matcher wins, and gains the point when the matcher loses. In this way, the Matching Pennies Game is a zero-sum game, whereby one player's gain is another player's loss.

While the mechanics of the game are simple, the concept is elevated by the various strategies which independent players develop to maximise their payoffs. This has led to the game being used as a way to investigate mixed strategies, direct competition, and decision making in tandem. The repetitive nature of the game, which is played across multiple trials, enables us to capture the many small decisions which accumulate into observable strategies. Such strategies have been replicated across diverse populations (Brosnan., 2018), including children (de Weerd et al., 2018), primates (Watzek et el., 2018), pigeons (Sanabria., & Thrailkill., 2009), and neural networks (Dolgova & Bartsev., 2019). They are further enforced by the verbal accounts collected from human players (Rutström & Wilcox., 2009). With literature on the Matching Pennies Game stretching back to the early 20th century (Pęski., & Szentes., 1913), the concept's simplicity has not detracted from the immense value it has in telling us more about the cognitive processes which underlie competitive decision making.

Assignment Description

In this assignment, we will explore two strategies commonly adopted in the Matching Pennies Game. This will be done firstly by considering the strategy as a verbal model, and secondly developing it from a verbal to formal model by implementing the strategies in algorithmic code. The following sections will describe each strategy in turn, with particular

emphasis on the cognitive constraints the player may experience in adopting the strategy. The reader is then encouraged to explore the R-markdown "Assignment_1_Script.Rmd", where each strategy has been implemented as an algorithm and visualised.

Part 1: Description of two strategies used to play the Matching Pennies Game
Part 2: Formalisation of the two strategies using plots and code
Part 3: Putting the strategies to the test in a series of battles

Part 1: Verbal Models

Strategy One: The Copycat

A player following our first strategy has been coined ‘the copycat’. Their strategy is to mimic the opponents game by always playing what their opponent last chose. Thus, if the opponent played *heads* in the last round, the copycat will play *heads* in the upcoming round.

This strategy balances upon one critical assumption, which flips in polarity depending on whether the copycat is in the matcher or mismatcher role. When the copycat is the matcher, they assume their opponent will continuously play the same side of the coin. If this were to hold true, the copycat could be sure to always win as they would simply copy and play that same side of the coin too. When the copycat is the mismatcher, they instead assume their opponent will always alternate what they last played. Thus, the copycat will win by simply playing what the opponent played before. In this way, the copycat is pinning their strategy to the role they currently hold and the assumption they make within that role. This leaves little room for flexibility or adaptation if the opponent does not behave in the expected way, but could lead to an almost perfect pay-off if their assumption holds true.

Another assumption of the copycat is that their opponent is blind to the mimicking strategy, or is making decisions independent of what the copycat plays. This seems paradoxical for a player who has built their own strategy upon feedback, but does not account for their opponents response in the design of their strategy. Indeed, this could be dangerous because the copycat strategy would be an easy one for the opponent to detect. The copycat’s moves are consistent and require limited memory processing for the opponent, which makes it predictable. As strategy predictability is a key weakness within the Matching Pennies Game, this could quickly send the copycat into a steep decline if their strategy is detected. They may be lucky and the opponent does not detect the copycat, but their success still depends upon the copycat’s compatibility with the strategy choice of the opponent.

These assumptions make the copycat a high risk strategy with the possibility to land almost anywhere on the pay-off scale. A copycat is vulnerable to easy detection, can not adapt to their opponents strategy, and is highly dependent upon the compatibility and consistency of

the opponent's game strategy. However, if we consider the strategy in terms of cognitive constraints, we see some greater appeal to the copycat.

A copycat is only required to remember one trial back and does not seem to theorise on what the opponent is doing beyond this previous trial. If we compare this to other strategies such as reinforcement learning, it is much less cognitively demanding in terms of memory and critical thinking. This makes it more robust against cognitive constraints such as fatigue or player error. Nevertheless, player error could in fact play to the copycat's favour, if the opponent is not behaving in a way consistent with their assumptions. Furthermore, a copycat is unaffected by the emotional constraints of winning or losing, as they do not adapt their strategy to the outcome. This would make them less susceptible to making irrational decisions in the heat of the moment and perhaps more reliable players. Reliability, however, is not necessarily a positive thing in the Matching Pennies Game as it makes one more predictable and thus more vulnerable.

While it is nice to theorise the copycat's strengths and weaknesses, our formal model should help to quantify how sensible the strategy really is for a player, and under which conditions.

Strategy 2: Win-Stay-Lose-Shift

A player following our second strategy has been coined a 'Win-Stay-Lose-Shifter' (WSLSer). Their strategy is to keep playing the same side of the coin when they are winning (i.e., stay) and flip to play the other side of the coin when the original side starts to lose (i.e., shift). Thus, if they won using *tails* in the previous round, they will continue to play *tails* in the preceding rounds until they encounter a loss, at which point they will shift to play *heads*.

A WSLSer is able to introduce some flexibility to their strategy, as they can choose how many trials it takes for them to lose before they shift to the other side of the coin. This may be something which is varied across trials, sometimes shifting after one loss and other times shifting after 3 losses. This would make their strategy more difficult for the opponent to detect. Alternatively, it may be something which is fixed, so that the player shifts consistently

after losing n-trials, with the most simple version being that they shift every time they face a loss. We will discuss the cognitive constraints of each further below.

Unlike the copycat, the WSLSer works with a different set of assumptions, pinning their decision upon the outcome feedback rather than the direct choice of an opponent in the previous round. Albeit, this is still an indirect response to the opponents previous choice, as the WSLSer decides to move based on whether their opponent is winning. This strategy comes with the parallel assumption that the opponent will play in sequences or runs, as opposed to alternating between sides with each round. These assumptions remain consistent, no matter whether the player is in the role of the matcher or the mismatcher.

Theoretically, such a feedback focus should make the WSLSer more adaptive to the various strategies played by their opponents, as they move in line with wins and losses. Thus, the approach could be seen as more strategic or calculated from a maximising perspective than the copycat's rigid mimicking rule. Therefore, the WSLS strategy could seem more compelling as it allows the agent to move with the dynamics of the game. However, this adaptiveness comes at the cost of cognitive constraints. The WSLS agent is required to keep track of their own previous move, their feedback outcome of n-trials back, and perhaps even learn the response behaviour of their opponent. Such complexity requires both critical thinking skills and a good memory. We know that working memory is limited and varies greatly between individuals (Unsworth., & Engle., 2007), so the need to rely on this for the WSLS strategy heightens the chance of human error and noise within the model. Both of these variables increase with the complexity of the decision, which depends on how many trials back the agent is required to remember and how complex their decision making function is (i.e., whether it is probabilistic or follows a strict rule of shifting after n-trials). An agent who must remember a large number of trials back (i.e., more than the typical working memory boundary of 3-5 trials (Cowan N., 2010), and who follows a stepped probabilistic function which increases the probability of shifting after each loss would face significantly more cognitive constraints than a copycat, having much more to remember and consider in their decision. Over time, these constraints may affect their performance.

Due to these constraints, one can imagine that a WSLS agent may benefit from keeping an open mind to change or exploration within their strategy. This could be by having an imperfect probability function which operated with a consistent rate no matter how many

losses had passed, thereby reducing the need for keeping track of how many losses there had been. Nevertheless, while reducing the cognitive constraints and perhaps making the WSLS strategy more difficult for the opponent to detect, we should be aware that this runs the risk of weakening the strategy's effectiveness as it could lead to the agent waiting too long to shift without realising how many losses they had endured.

Another key consideration in the WSLS strategy is the emotional element introduced by responding to wins and losses. Humans are not always rational decision makers, and this is particularly true when they have the emotional pressure of gaining or losing something (Lerner et al., 2015). The loss-aversion bias, whereby agents have a greater tendency to avoid loss than seek a gain (Johnson et al., 2006), is a well replicated example of how emotions can affect decisions and it is likely the same effect could be found throughout the WSLS strategy.

To conclude, the WSLS strategy has compelling features such as being flexible and adaptive yet may be greatly dependent upon individual factors such as the agent's own working memory capacity and their tendency to respond to emotional situations. This is hard to code into a formal model, but can be partly accounted for by introducing a greater element of noise into the WSLS agents. In our model, we will design an adaptive WSLSer, who will vary the number of trials between losses before it shifts to the other side, by using a stepped probabilistic function with a memory of 5 trials. These agents can then be played against the vanilla WSLSers who switch every time they face a loss.

With our agents now verbalised, we can put them to the test by using simulated agents.

Part 2: From Verbal to Formal Models

The formalisation of our verbal models rests upon three fundamental principles, which will all be implemented in RStudio (version 4.1.2 'Bird Hippie') using the *'tidyverse'* package (Wickham et al., 2019). The first principle relates to the binary nature of the model. As there are two possible outcomes in the Matching Pennies Game (a *'heads'* or *'tails'*), we use a binomial probability function to generate our agent's choice in each strategy. In other words, we can say that a 1 reflects an agent selecting *'heads'* and a 0 reflects an agent selecting *'tails'*. This can be demonstrated by using an agent operating with a random choice strategy:

Let us assume there is a random agent who has a 50% chance of selecting either a *'heads'* or *'tails'*. To simulate this, we simply use the 'rbinom()' package which takes 3 parameters of *'n'*, *'size'*, and *'prob'* to select either a 0 (*tails*) or 1 (*heads*) for each trial. These can be thought of as the numerical range, the number of trials, and the rate of probability of getting *'1'* respectively. The selections are saved into a variable named *'RandomAgent'*:

```
RandomAgent <- rbinom(n = trials, size = 1, prob = 0.5)
```

Code block: 1

The agent's choices can be plotted by recording the selection for each trial into a dataframe and then using the *'ggplot'* package (Wickham., 2016). *Figure 1* below demonstrates this with a random agent operating with a non-biased rate (0.5), a rate biassed towards 1 (0.7) , and a rate biassed towards 0 (0.3):

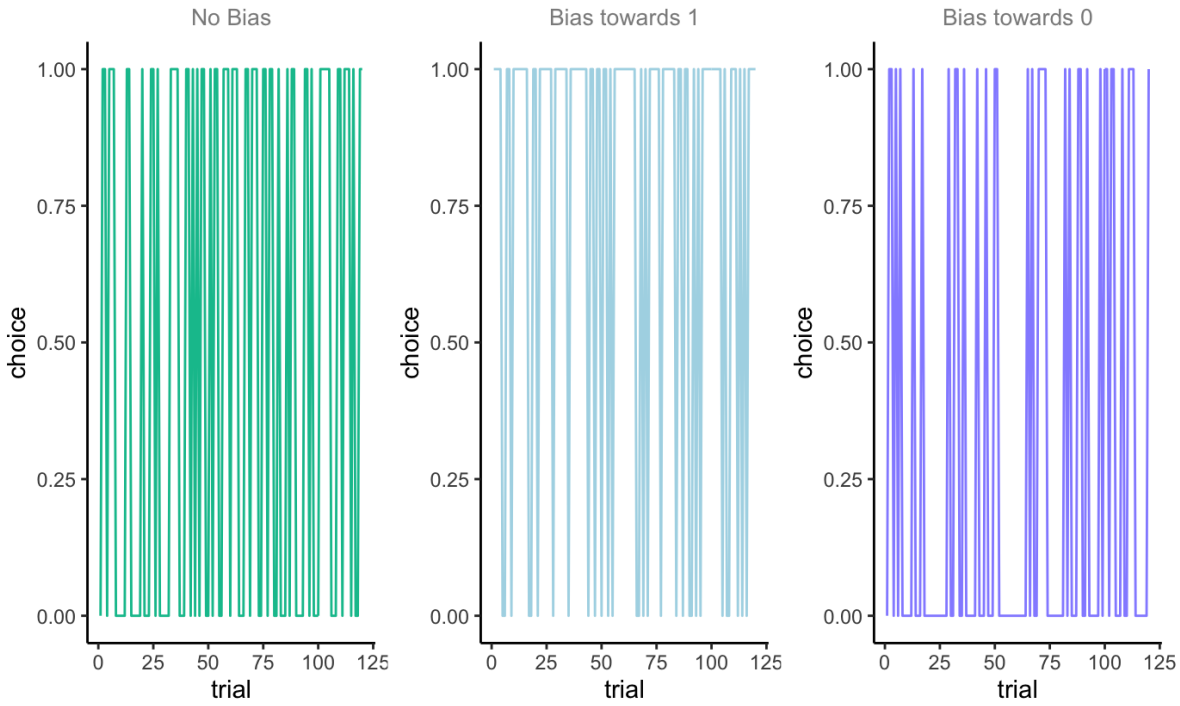


Figure 1: Line graphs showing the binomial choice of random agents across 120 trials

The second principle relates to an agent's bias. In a perfect world, humans (or random agents) would behave rationally and without implicit bias. As cognitive scientists however, we know that this is most likely not the case and it is more probable that individual agents behave with some level of natural bias (Dyson et al., 2020). We can visualise how a bias within the model may look by plotting the agent's cumulative rate across the number of trials they play. A non-biassed agent could be expected to have a cumulative rate hovering around 0.5, meaning they are choosing 0 and 1 in equal proportions, while a biassed agent towards 1 could be expected to have a higher cumulative rate and a biassed towards 0 could be expected to have a lower cumulative rate. This is visualised below using the same rates as in *Figure 1:*

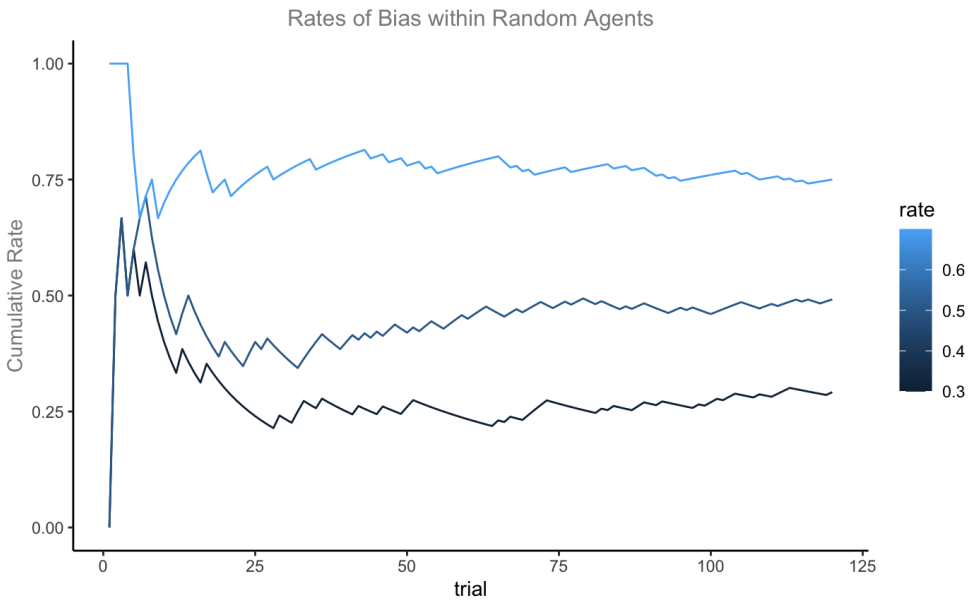


Figure 2: Plot showing the 'Bias towards 1' agent with rate of 0.7 (top), 'No bias' agent with rate of 0.5 (middle), and 'Bias towards 0' agent with rate of 0.3 (bottom) across 120 trials.

To build robust models, we should therefore be aware of this and allow for biases to be accounted for within our cognitive model. This can be done by creating an amendable *'rate'* parameter which allows the user to vary or select how biassed their agents are. A suggestion of how a user may vary rates across a number of agents is shown below:

```
# Number of agents
Agents <- 100
# Vary rates between 0.5 to 1, in intervals of 0.05
for (rate in seq(from=0.5, to=1, by=0.05)){
      for (agent in seq(agents)){

[**insert strategy function **]

}}
```

Code block: 2

For the purposes of this assignment, random agents will be assigned a bias rate of 0.7, meaning they have a 70% chance of selecting *'heads'* or *'1'*.

Finally, our third principle relates to the tendency of non-virtual agents to make mistakes. Here, non-virtual refers to agents which are not run by a computer program such as humans, primates, or whatever live creature is playing the Matching Pennies Game. A distinction is necessary to take our model from being a generative model to a cognitive model, whereby the

latter requires us to account for how humans actually perceive and process information, which is prone to error and mistakes (Dyson et al., 2020). If we want our model to truly reflect cognitive behaviour, we need to pre-empt the occurrence of error by including a noise parameter which can reflect an X% error rate. This can be thought of as representing all the mistakes a non-virtual agent can be expected to make such as wrong button presses, a lapse of attention, an irrational emotion-driven response, or memory defects.

To define a noise parameter within our model, we build an additional clause into the random agent's behaviour which can be used to replace their original choice at a rate equivalent to the noise parameter. This clause samples from a binomial distribution with each choice made, using the noise parameter (i.e., the error rate) as the probability of returning a *'1'*. If a *'1'* is returned, the agent's original choice is re-sampled from a new binomial distribution with a rate of 0.5. To further exemplify this, if we were to set an error rate of 0.1, then we could expect that around 10% of the time the agent's choice would not be their original rational decision, but instead a new random choice sampled from a non-biassed binomial distribution. The code below outlines this process:

```
# Parameters
# Creating an agent with a 70% bias towards 1, and error rate of 0.1
noise <- 0.1
rate <- 0.7

# Create function to make agent using parameters
RandomAgent_with_noise <- function(input, rate, noise) {
      # let n be the length of the input data
      n <- length(input)

      # make a choice using the rate as the probability of a 1
      choice <- rbinom(n, 1, rate)

      # also sample from a random distribution
      # if the outcome is 1, resample 'choice'
      if (rbinom(1, 1, noise) == 1) {choice <- rbinom(1, 1, 0.5)}

      # return the agents choice
      return(choice) }
```

Code block: 3

Having addressed the fundamental principles within our model, we can turn our attention to how we can algorithmically formulate the four types of agents used throughout the assignment.

Random Agents

A random agent picks *'heads'* or *'tails'* independent of what their opponent is doing. As mentioned above, they can have a pre-defined implicit bias and an error rate (noise). In this simulation, our random agent's will have a bias of 70% leaning towards heads and we will give them a conservative error rate of 10%. Future versions of the simulation could experiment with varying these parameters as suggested in '*Code block 3*'. The function used to generate our random agents is defined below:

```
# parameters
noise <- 0.1
rate <- 0.7
# random agent function
RandomAgent <- function (input, rate, noise) {
      n <- length(input)
      choice <- rbinom (n, 1, rate)

      if (rbinom(1, 1, noise) == 1) {choice = rbinom(1, 1, 0.5)}
      return(choice) }
```

Code block: 4

Win-Stay-Lose-Shift-Basic Agents

A Win-Stay-Lose-Shift-Basic (WSLSB) agent responds to the outcome of the previous round, by staying with the previous choice if they won the point, and immediately switching to the other side if they lost the point. Thus, their performance in the previous round determines their choice for the next simulation. We can formulate this by creating a 'feedback' variable, which is used in an if statement to say whether an agent should stay with the previous choice, or shift to the other one. The binomial nature of the Matching Pennies Game allows us to use simple maths to control this flow (using 1 - prevChoice) and again an error rate of 10% is built into the agents to reflect their erroneous nature. The function used to generate WSLSB agents is defined below:

```
# parameters
noise <- 0.1

# WSLSB agent function
# Takes in the previous choice and feedback (loss (0) or win (1))
WSLSB_Agent <- function(prevChoice, Feedback){
      if (Feedback == 1) {  # if agent won (ie Feedback = 1), stay.
      choice = prevChoice }

      else if (Feedback == 0) { # if agent lost(ie Feedback = 0), shift.
      choice = 1 - prevChoice  }  # this ensures outcome is opposite

      if (rbinom(1, 1, noise) == 1) {choice <- rbinom(1, 1, 0,5) }
      return (choice) }
```

Code block: 5

Copycat Agents

A copycat picks the previous choice of their opponent, independent of whether they won or lost the point. They operate without a rate of bias as they are not making autonomous decisions, however they will inherit the natural bias of their opponent. Like all other agents, the copycat is also susceptible to errors and so will be given the same error rate of 10%. The function used to generate copycat agents is defined below:

```
# parameters
noise <-  0.1

# copycat agent function
CopyCat <- function(oppChoice, noise) {
      If (oppChoice == 1) {     # if opponent selects heads
              choice = 1}       # make the copycat's next move heads
      else if (oppChoice == 0) {
                  choice = 0}  # otherwise, make the next move tails

      If (rbinom(1, 1, noise) == 1) { choice <- rbinom(1, 1, 0.5)}
      return (choice) }
```

Code block: 6

Win-Stay-Lose-Shift-Probabilistic Agents

A Win-Stay-Lose-Shift-Probabilistic (WSLSP) agent operates in a similar fashion to the WSLSB agents, with the one difference being that they vary the number of losses they can endure before shifting. This can be between 1 and 5 losses, with the agent always shifting if they manage to reach 5 consecutive losses without moving. Their decision whether to '*shift'* or *'stay'* after each loss is controlled using a stepped probabilistic function, which increases the likelihood that they'll shift with each new loss they encounter. Thus, the more consecutive losses they face, the more likely they are to shift. For this reason, WSLSP agents require a greater memory capacity than WSLSB or copycat's, as they need to remember their own moves up to 5 trials back. Whenever they enjoy a win or shift to play a new side of the coin, their memory is reset to 0. Again, like the other agents, the WSLSP agents also operate with an error rate, however this is 15% as their cognitive constraints mean they are slightly more likely to make errors than agents following simpler strategies.

First, we define the probabilistic function which determines whether the agent *'shifts'* or *'stays'* after encountering a loss. This function is outlined below:

```
# - Probabilistic Function -

# If an agent gets a 1, they shift. If they get a 0, they stay.
# As we want our agents to move, but not every time, we set low rates of
probability at each step
check_memory <- function(memory, prevChoice) {
      if (memory == 1) {
      # select a 0 or 1, with a 10% chance of getting a 1
      selection <- rbinom(n = 1, size = 1, prob = 0.1) {
            # if they select a 1, they shift and memory is reset to 0
            if (selection ==1) {
                  choice <- 1 - prevChoice
                  memory = 0 }
            # if they select a 0, they stay and memory remains the same
            else if (selection == 0) {
                  choice <- prevChoice
                  memory <- memory }
            }
      else if (memory == 2){
      #select a 0 or 1, with a 30% chance of getting a 1
      selection <- rbinom(n = 1, size =1, prob = 0.3)
```

```
             if (selection == 1){
                        choice <- 1 - prevChoice
                        memory = 0 }
             else if (selection == 0){
                        choice <- prevChoice
                   memory <- memory }
             }
      else if (memory == 3){
      #select a 0 or 1, with a 50% chance of getting a 1
      selection <- rbinom(n = 1, size =1, prob = 0.5)
             if (selection == 1){
                        choice <- 1 - prevChoice
                        memory = 0 }
             else if (selection == 0){
                        choice <- prevChoice
                   memory <- memory }

}
      else if (memory == 4){
      #select a 0 or 1, with a 70% chance of getting a 1
      selection <- rbinom(n = 1, size =1, prob = 0.7)
             if (selection == 1){
                        choice <- 1 - prevChoice
                        memory = 0 }
             else if (selection == 0){
                        choice <- prevChoice
                   memory <- memory }
}
      else if (memory == 5){
      # agents always move when they get to 5 consecutive losses
             choice <- 1 - prevChoice
             Memory <- 0 } # after moving we set memory back to 0

      # R functions only return 1 output,
      # So, we save our 2 variables into a vector which can be
      # indexed to access the variables (1 = memory, 2 = choice).
      selection_variables <- c(memory, choice)
      return(selection_variables)
}
```

Code block: 7

Second, we create our WSLSP agent function which uses ‘check_memory’ from above:

```
# parameters
noise <- 0.15
```

```
# WSLSP agent function
WSLSP_Agent <- function(prevChoice, memory, Feedback){
# if the agent wins
      if (Feedback == 1) {
     # reset their memory
        memory = 0
    # set their choice to their own prevChoice
      choice = prevChoice
# If the agent loses
else if (Feedback == 0) {
     # add one to the memory count
        memory <- memory + 1
     # determine their next move using their memory
        agent_behaviour <- check_memory(memory, prevChoice)
    # extract variables from the 'agent_behaviour' vector
   #  1 = memory, 2 = choice
        memory <- agent_behaviour[1]
        choice  <- agent_behaviour[2]
# Noise function
if (rbinom(1, 1, noise) == 1) { choice <- rbinom( 1, 1, 0.5)}

#combine memory and choice into a list
list <- c(memory, choice)
return(list)
}
```

Code block: 8

With our agents constructed, we can put them to the test through a series of planned battles to see which strategy fares best and whether this stays consistent against the different types of opponents.

Part 3: Battles of the Strategies

Our key interest throughout this assignment is to understand more about our two defined strategies of CopyCat and the Win-Stay-Lose-Shift-Probabilistic (WSLSP). This goes beyond understanding how they are verbalised and formalised into code, but requires an understanding of how they fare when played against each other. This is where the real magic arises and can tell us more about the robustness of each strategy when tested under controlled conditions. If we are lucky, it may also be able to reveal something about the cognitive processes underlying each strategy and how these play into or compliment each other.

The random agents and WSLSB agents are used as controls, to test each strategy individually and provide a baseline for comparison. After creating this foundation, we can play our two strategies against each other and see which, if either, comes out on top. A discussion will then try to provide some interpretation of our findings and see if we can extract any cognitive insights from the battles.

To maintain a controlled environment, we will create 100 agents for each side of the battle and allow them to compete across 120 trials. A winner is defined as the one with the most points at the end. For simplicity, we will not vary our random agents nor our WSLSB agents, however further extensions of the topic could explore these two strategies on their own. The following schedule has been constructed for the battles whereby the main role is to be the matcher:

	Matcher	**Mis-Matcher**
Battle 1	CopyCat	Random
Battle 2	CopyCat	WSLSB
Battle 3	WSLSP	Random
Battle 4	WSLSP	WSLSB
– Evaluation and final battles –		
Battle 5	CopyCat	WSLSP
Battle 6	WSLSP	CopyCat

Table 1: Schedule for the Matching Pennies Tournament

Without further ado, we can set our battles into action and compare their results. The code for each of these can be found in the RMarkdown file found within the GitHub repository.

Results

	Matcher	Mis-Matcher
Battle 1	CopyCat (57%)	Random (43%)
Battle 2	CopyCat (49%)	WSLSB (51%)
Battle 3	WSLSP (57%)	Random (43%)
Battle 4	WSLSP (90%)	WSLSB (10%)
– Evaluation and final battles –		
Battle 5	CopyCat (82%)	WSLSP (18%)
Battle 6	WSLSP (90%)	CopyCat (10%)

Table 2: Results from the Matching Pennies Tournament

Evaluating the CopyCat Strategy

The Copycat strategy fared well when in the matcher's role. It performed at a higher rate than the random agents, around the same rate as the WSLSB agents, and significantly better than the WSLSP agent. However, this fate reversed while it was put into the mismatchers role, as we see in 'Battle 6', where it was unable to respond to any feedback and continued to play the same side despite losing consecutively. This demonstrates the danger of choosing such a rigid strategy, which considers neither the feedback from the previous round nor the strategy of the other player. In particular, the CopyCat is a risky strategy against other rigid roles such as a win-stay-lose-shift opponent, where they could acquire a cumulative rate of 0% (or 100%), if neither of the agent sets has flexibility or a noise clause accounted for in their computations. We can see this play out across two scenarios below if we consider the CopyCat in the mismatcher role, playing a WSLSB agent. Here, the fate of CopyCat lies completely in whether the first trial was a match or mismatch:

CopyCat (mismatcher) 'v' WSLSB (matcher)

Trial	CopyCat	WSLSB	Outcome	Winner		CopyCat	WSLSB	Outcome	Winner
1	*heads*	*heads*	match	WSLSB		*heads*	*tails*	mismatch	CopyCat
2	*heads*	*heads*	match	WSLSB		*tails*	*heads*	mismatch	CopyCat
3	*heads*	*heads*	match	WSLSB		*heads*	*tails*	mismatch	CopyCat

4	*heads*	*heads*	match	WSLSB		*tails*	*heads*	mismatch	CopyCat
5	*heads*	*heads*	match	WSLSB		*heads*	*tails*	mismatch	CopyCat

Table 3: Demonstrating the game dynamics across 5 trials between CopyCat and WSLSB agents.

To understand the dynamics of the CopyCat strategy better, we can plot the cumulative rate which occurs across the game as the copycat agent is in the matcher and mismatcher roles:

Copycat as Matcher:

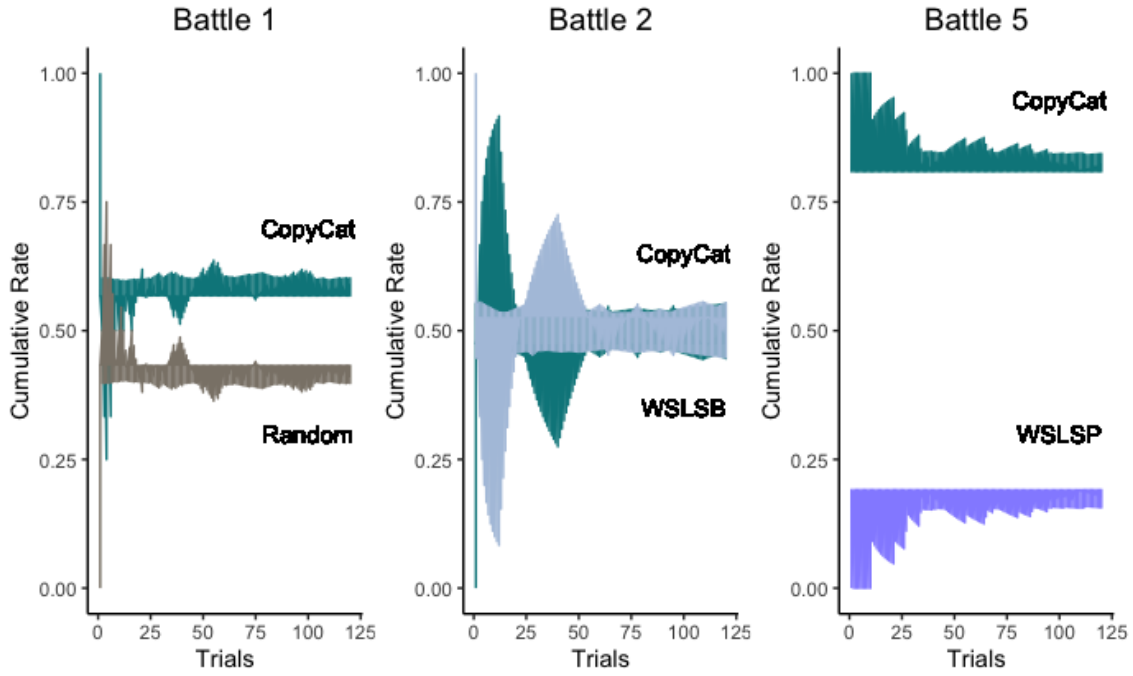


Figure 3: Plot showing the cumulative rate of CopyCat agents in the matcher role against random agents (Battle 1), WSLSB agents (Battle 2), and WSLSP agents (Battle 5), across 120 trials.

Copycat as Mis-Matcher:

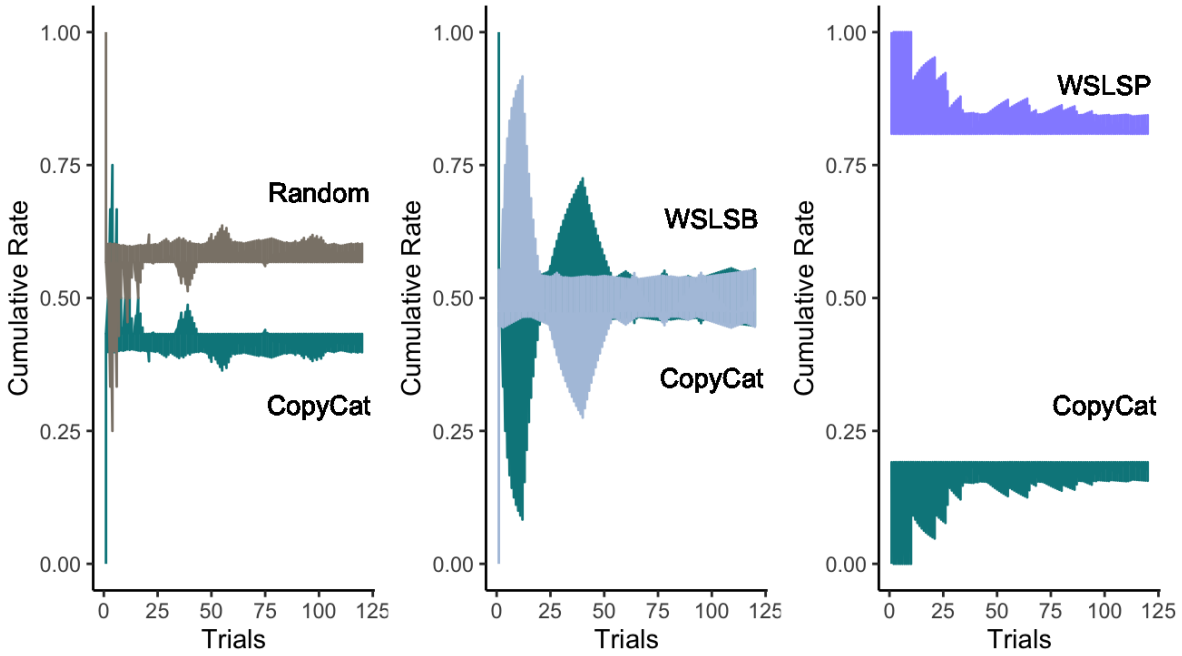


Figure 4: Plot showing the cumulative rate of CopyCat agents in the mis-matcher role against random agents, WSLSB agents, and WSLSP agents (Battle 6), across 120 trials.

Evaluating the WSLSP Strategy

Similarly, the WSLSP strategy thrives in the matcher role. It performs at a notably higher rate when playing both an agent similar to itself (WSLSB) and the CopyCat, and maintains a moderately higher rate over the random agents. When placed in the mismatcher role however, it struggles to adapt to the opponents strategy and suffers heavy losses. This is seen in Battle 5, where the WSLSP agent is forced to switch frequently to avoid a string of matches as the CopyCat mimics WSLSP's choices. In this scenario, the delayed probabilistic function and noise clause play directly into the CopyCat's hands as the WSLSP agent's decision not to shift after every loss increases the chance of matching, thereby allowing the CopyCat to swoop in and collect some certain wins (i.e., matches). Any time the WSLSP agent delays a shift or the noise clause stops them from moving, it is free points for the CopyCat. The easiest way to make sense of this scenarios is to walk through five trials:

Scenario 1: WSLSP as the Mis-matcher

	WSLSP (mis-matcher)	CopyCat (matcher)	Outcome	Agent that wins point	WSLSP decision for next round
Trial 1	*heads*	*heads*	match	CopyCat	*Switch*
Trial 2	*tails*	*heads*	mismatch	WSLSP	*Stay*
Trial 3	*tails*	*tails*	match	CopyCat	*Stay*
Trial 4	*tails*	*tails*	match	CopyCat	*Switch*
Trial 5	*heads*	*tails*	mismatch	WSLSP	*Stay*

Table 4: Demonstrating game dynamics between WSLSP (mismatcher) and CopyCat

While the probabilistic function works against the WSLSP agent in the mismatcher role, it provides the agent with more protection than the rigid WSLSB agent overall. If we consider the scenario outlined in *Table 3*, whereby the first result was able to determine the entire game, we can see the value of having a probabilistic function which introduces more uncertainty into the turn-taking. For example, the function works to the WSLSP agent's favour when in the matcher role and the noise clause knocks the game out of sync. Here we can imagine a scenario whereby the WSLSP agent is winning, but the noise clause forces them to switch sides. The WSLSP agent encounters a loss and switches back, however the CopyCat playing 1 step behind is mimicking these switches and collecting the points as they are no longer matching. The WSLSP's probabilistic function now saves the agent from falling into the alternating win-loss rate we see in 'Battle 2', as the agent chooses not to switch after every loss and therefore quickly redirects itself back into a winning streak of matches. We can see this play out below:

Scenario 2: WSLSP as the Matcher

	WSLSP (matcher)	CopyCat (mis-matcher)	Outcome	Agent that wins point	WSLSP decision for next round
Trial 1	*heads*	*heads*	match	WSLSP	*Stay*
*****Noise clause triggered - flipping the decision*****					
Trial 2	*tails*	*heads*	mismatch	CopyCat	*Switch*
Trial 3	*heads*	*tails*	mismatch	CopyCat	***Stay*
Trial 4	*heads*	*heads*	match	WSLSP	*Stay*
Trial 5	*heads*	*heads*	match	WSLSP	*Stay*

Table 4: Demonstrating game dynamics between WSLSP (matcher) and CopyCat
*** probabilistic function forces agent to stay put for this loss*

The cumulative rates of the WSLSP agents are able to give us a better insight into how the agent performs against different strategies in each role across trials:

WSLSP as Matcher:

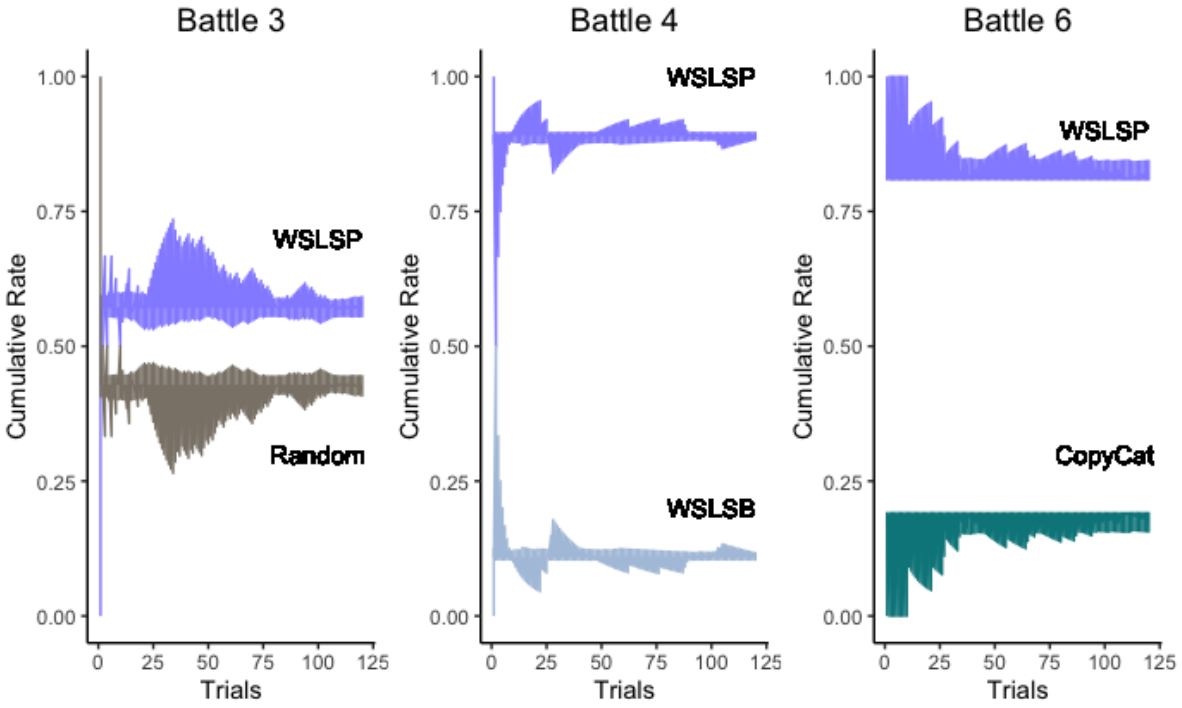


Figure 5: Plot showing the cumulative rate of WSLSP agents in the matcher role against random agents (Battle 3), WSLSB agents (Battle 4), and CopyCat agents (Battle 6), across 120 trials.

WSLSP as Mis-matcher:

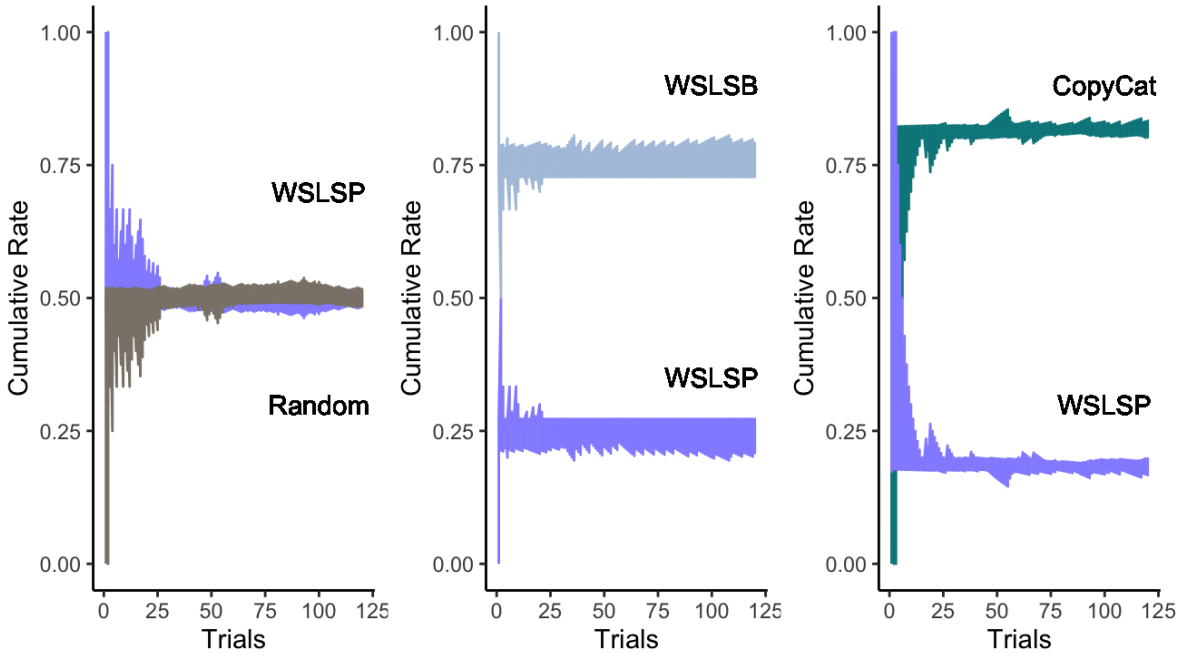


Figure 6: Plot showing the cumulative rate of WSLSP agents in the mis-matcher role against random agents, WSLSB agents, and CopyCat agents (Battle 5), across 120 trials.

Consideration of Cognitive Constraints

As aforementioned, a great appeal of the CopyCat strategy is that it is the least cognitively demanding and thus robust against cognitive constraints such as memory deficits or emotional responses. While this may be seen as a consistent game play (e.g., *Battle 1 & 5*), in reality the lack of response to emotional losses becomes more of a disadvantage to the agent (e.g., Battle 6). This may suggest that some level of emotional response is indeed beneficial to an agent, motivating them to adapt their strategy when the losses come. Further, the predictability of CopyCat would become a major issue as soon as they were to play a more 'switched on' agent such as a reinforcement learning agent.

Switching our attention to the more adaptive WSLSP agent, we do not see much benefit in performance despite the greater cognitive load this agent faces. *Battle 4* demonstrates the benefit of a good working memory to the WSLSP agent while in the matcher role against a WSLSB agent, however this same cognitive function becomes its downfall as soon as the roles switch and it plays the mismatcher (*Figure 6)*. This is in contrast to CopyCat who seems to be an even battle partner to the WSLSB agent no matter whether playing the matcher or mismatcher role. Thus it seems that memory can either be to one's advantage or disadvantage in the Matching Pennies Game. Nevertheless, the

WSLSP agent here is also unable to make an ‘emotional’ choice as they operate with a probabilistic function in response to losses and so it is difficult to claim much of how emotion drives an agent's performance. This is an area that could be explored in the upcoming assignments.

Please continue reading through assignment 2, 3, and 4, to dive deeper into Matching Pennies strategies and the dynamics which reinforcement learning can bring to such games.

References:

Brosnan, S. F. (2018). Insights into human cooperation from comparative economics. *Nature Human Behaviour*, *2*(7), 432-434.

Cowan, N. (2010). The magical mystery four: How is working memory capacity limited, and why?. *Current directions in psychological science*, *19*(1), 51-57.

Dyson, B. J., Musgrave, C., Rowe, C., & Sandhur, R. (2020). Behavioural and neural interactions between objective and subjective performance in a Matching Pennies game. *International Journal of Psychophysiology*, *147*, 128-136.

de Weerd, H., Diepgrond, D. & Verbrugge, R. (2018). Estimating the Use of Higher-Order Theory of Mind Using Computational Agents. *The B.E. Journal of Theoretical Economics*, *18*(2), 20160184. https://doi.org/10.1515/bejte-2016-0184

Dolgova, T., & Bartsev, S. (2019, May). Neural networks playing 'matching pennies' with each other: reproducibility of game dynamics. In *IOP Conference Series: Materials Science and Engineering* (Vol. 537, No. 4, p. 042002). IOP Publishing.

Johnson, E. J., Gächter, S., & Herrmann, A. (2006). Exploring the nature of loss aversion. *Available at SSRN 892336*.

Lerner, J. S., Li, Y., Valdesolo, P., & Kassam, K. S. (2015). Emotion and decision making. *Annual review of psychology*, *66*, 799-823.

Pęski, M., & Szentes, B. (1913). LOCAL STABILITY OF STATIONARY EQUILIBRIA (VERY PRELIMINARY AND VERY INCOMPLETE)

Rutström, E. E., & Wilcox, N. T. (2009). Stated beliefs versus inferred beliefs: A methodological inquiry and experimental test. *Games and Economic Behavior*, *67*(2), 616-632.

Sanabria, F., & Thrailkill, E. (2009). Pigeons (Columba livia) approach Nash equilibrium in experimental Matching Pennies competitions. *Journal of the experimental analysis of behavior*, *91*(2), 169-183.

Watzek, J., Smith, M. F., & Brosnan, S. F. (2018). Comparative economics: Using experimental economic paradigms to understand primate social decision-making. In *Evolution of primate social cognition* (pp. 129-141). Springer, Cham.

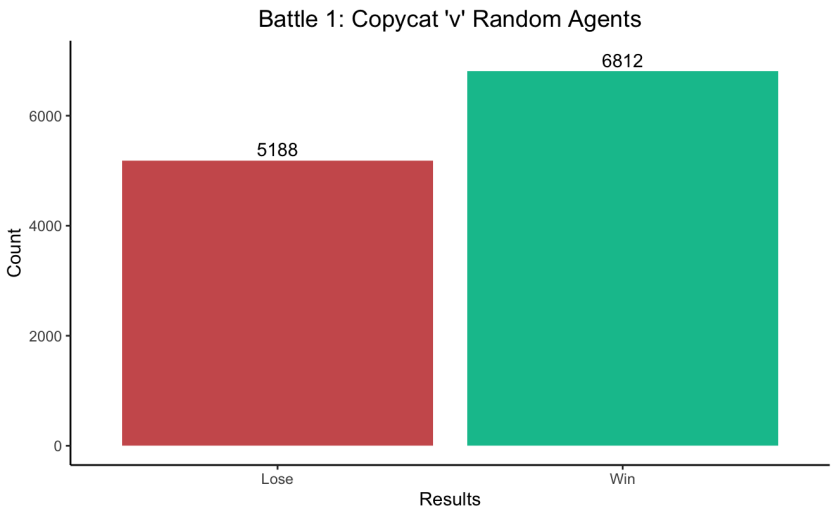
Wickham, H. (2016) ggplot2: Elegant Graphics for Data Analysis. *Springer-Verlag New York*. 978-3-319-24277-4 (https://ggplot2.tidyverse.org)

Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

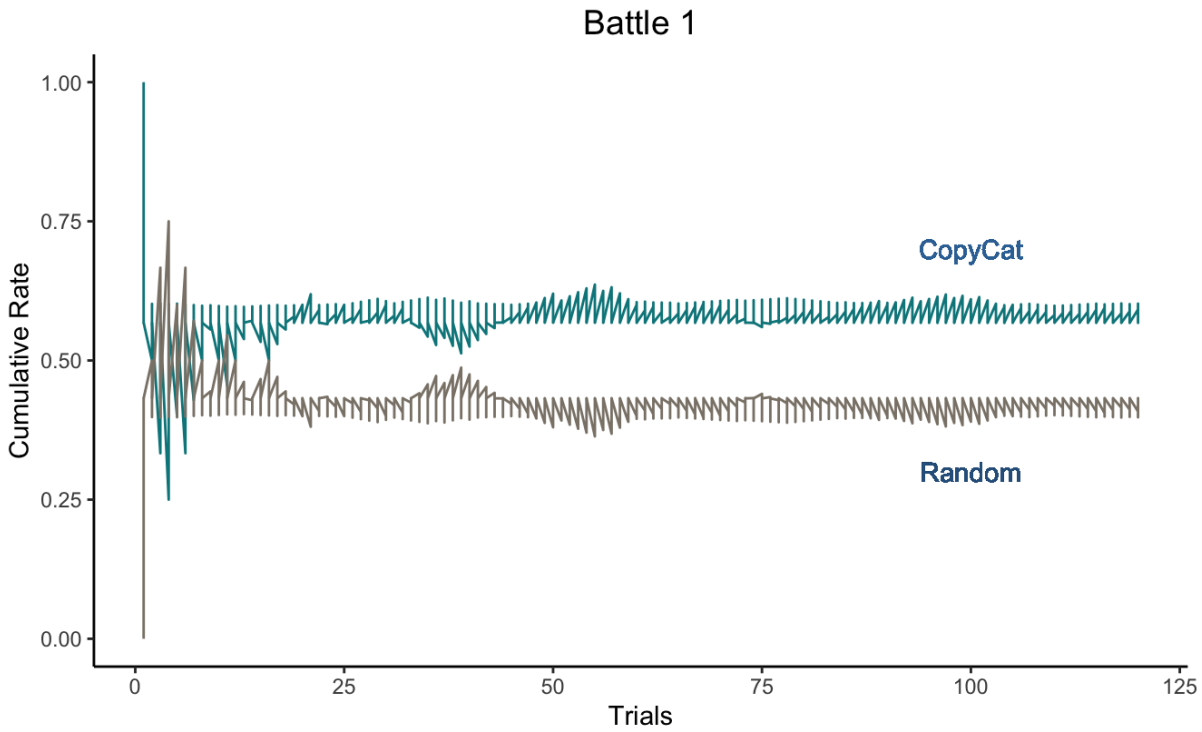
Unsworth, N., & Engle, R. W. (2007). The nature of individual differences in working memory capacity: active maintenance in primary memory and controlled search from secondary memory. *Psychological review*, *114*(1), 104.

Appendix

Battle 1: CopyCat ‘v’ Random Agents

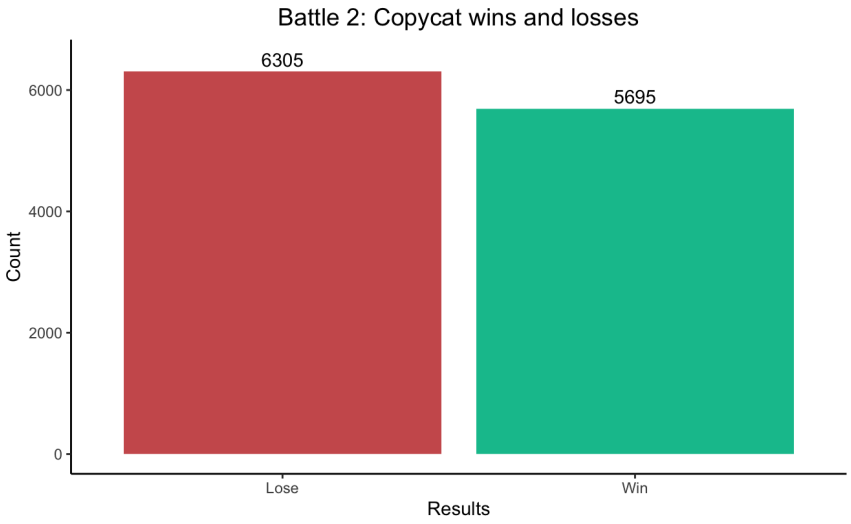


Appendix figure 1: Distribution of CopyCat’s wins and losses against random agents in Battle 1

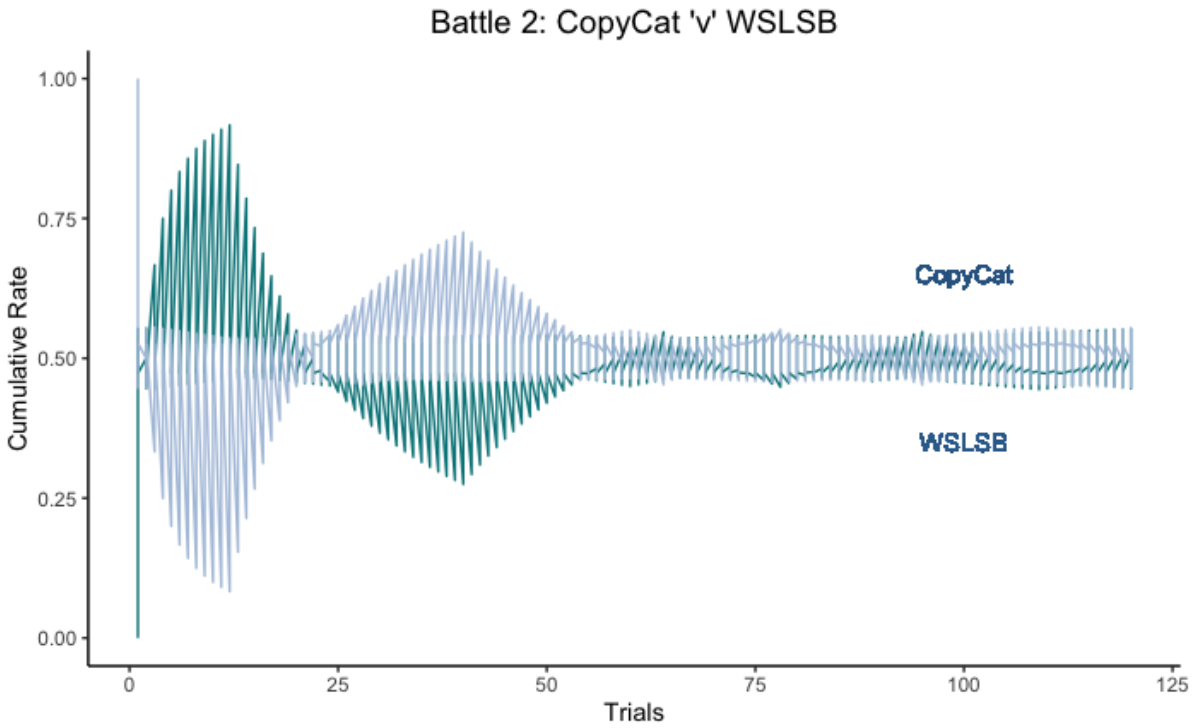


Appendix figure 2: Cumulative rates of CopyCat and Random agents across 120 trials in Battle 1. Copycat mean rate (0.576) with s.d. (0.01) and Random Agent mean rate (0.424) with s.d. (0.01)

Battle 2: CopyCat ‘v’ WSLSB

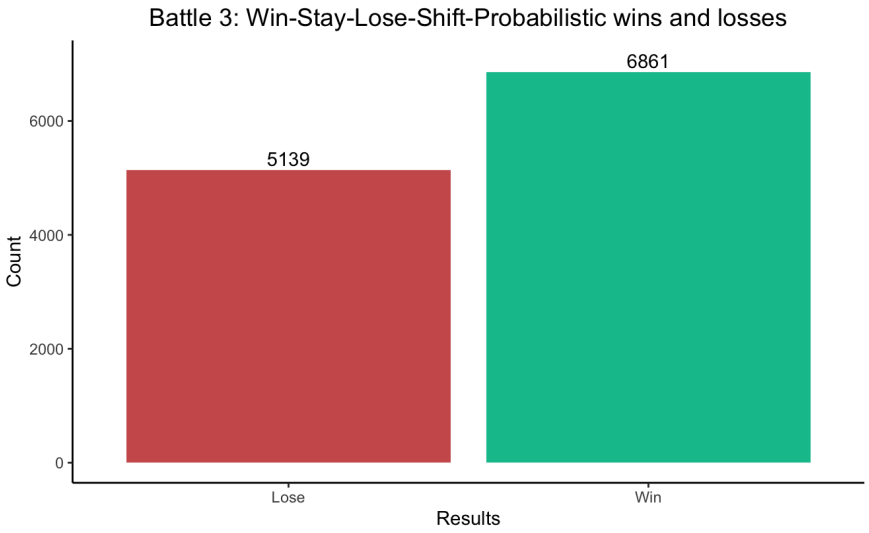


Appendix figure 3: Distribution of CopyCat’s wins and losses against WSLSB agents in Battle 2

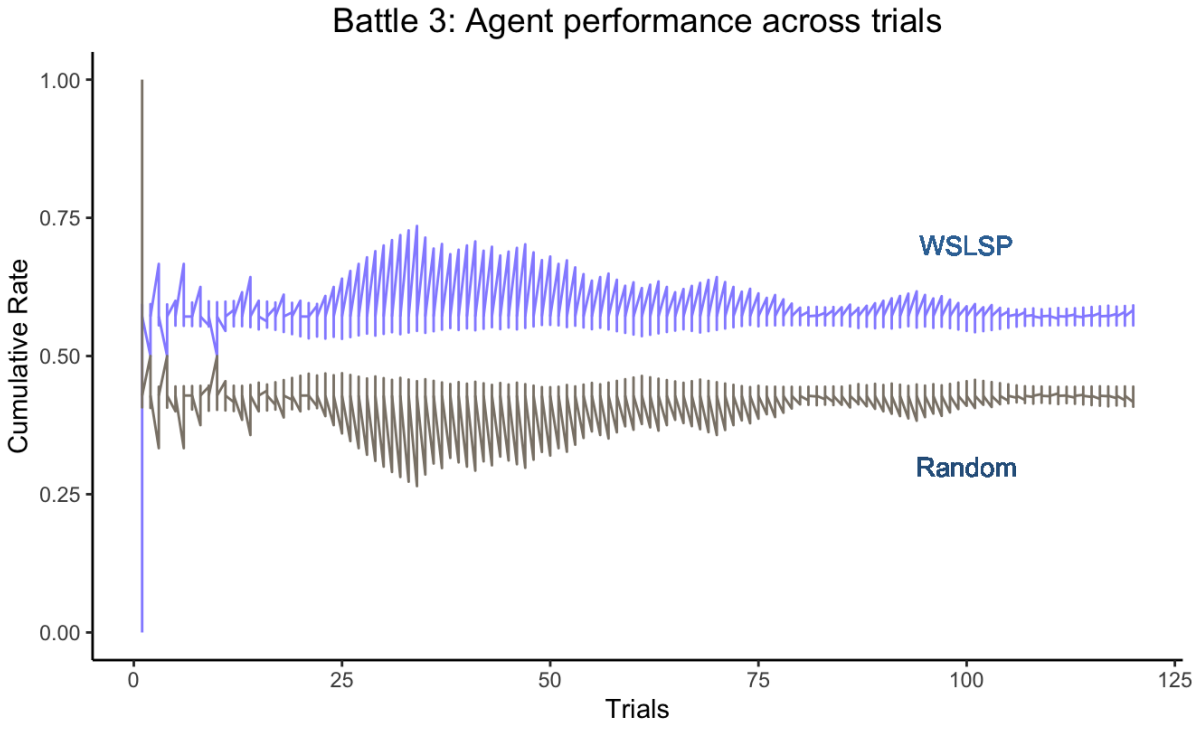


Appendix figure 4: Cumulative rates of CopyCat and WSLSB agents across 120 trials in Battle 2. Copycat mean rate (0.483) with s.d. (0.02) and WSLSB agent mean rate (0.516) with s.d. (0.02)

Battle 3: WSLSP ‘v’ Random Agents

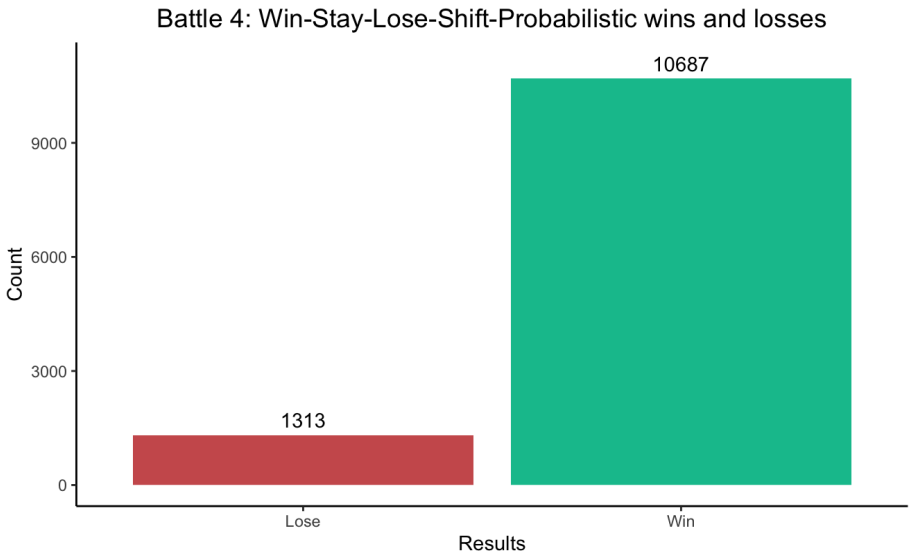


Appendix figure 5: Distribution of WSLSPs wins and losses against Random agents in Battle 3

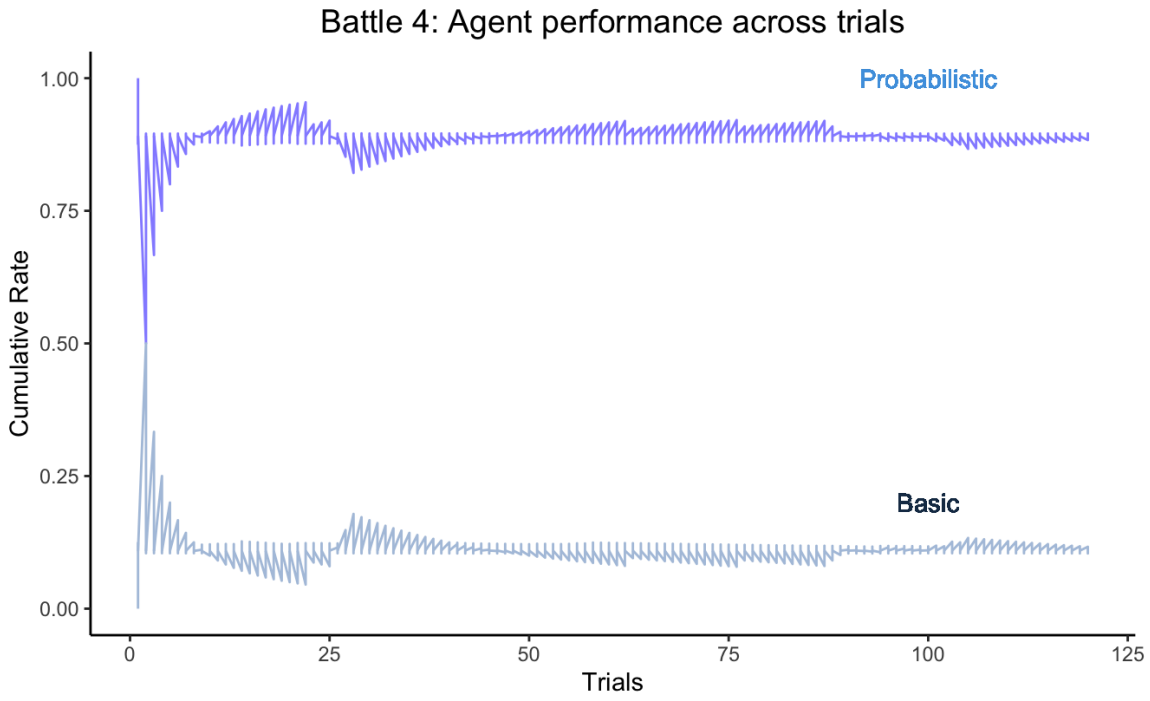


Appendix figure 6: Cumulative rates of WSLSP and Random agents across 120 trials in Battle 3. WSLSP mean rate (0.567) with s.d. (0.01) and Random agent mean rate (0.433) with s.d. (0.01)

Battle 4: WSLSP ‘v’ WSLSB

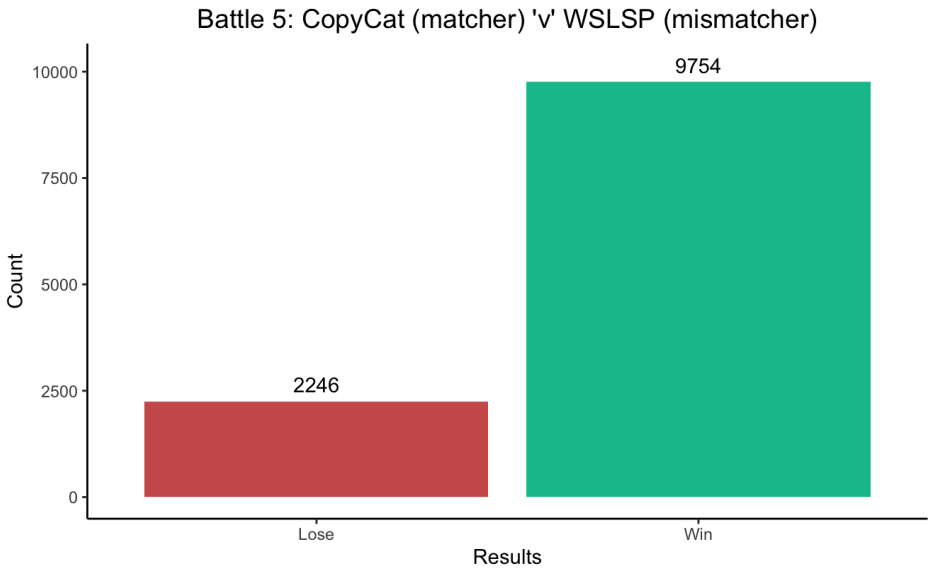


Appendix figure 7: Distribution of WSLSPs wins and losses against WSLSB agents in Battle 4

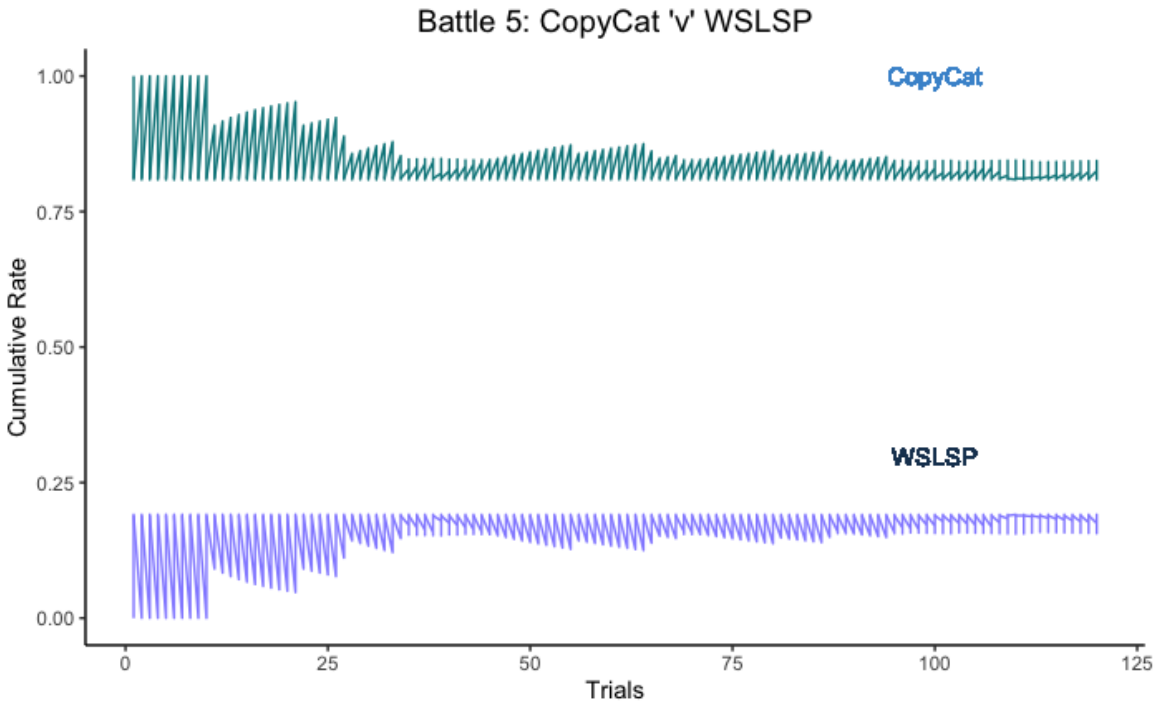


Appendix figure 8: Cumulative rates of WSLSP and WSLSB agents across 120 trials in Battle 4. WSLSP mean rate (0.891) with s.d. (0.01) and Random agent mean rate (0.109) with s.d. (0.01)

Battle 5: CopyCat (matcher) ‘v’ WSLSP (mis-matcher)

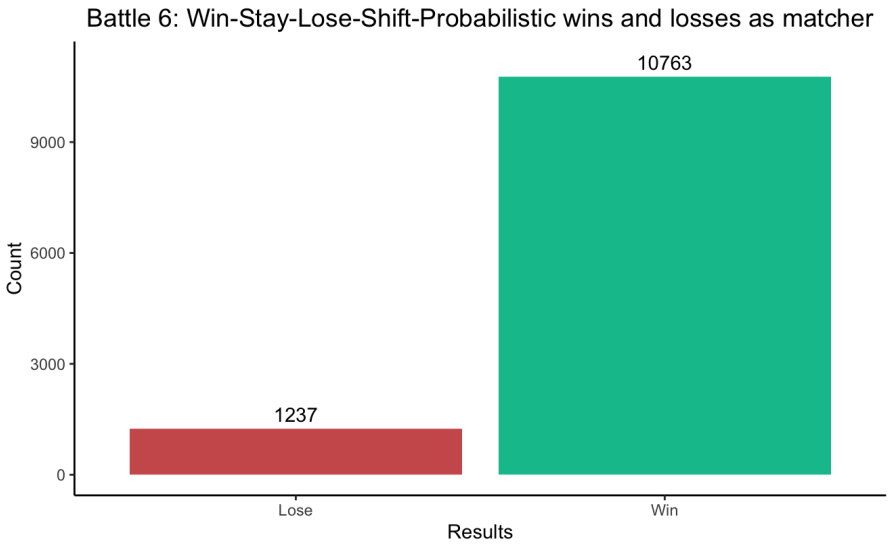


Appendix figure 9: Distribution of CopyCat’s wins and losses against WSLSP agents in Battle 5

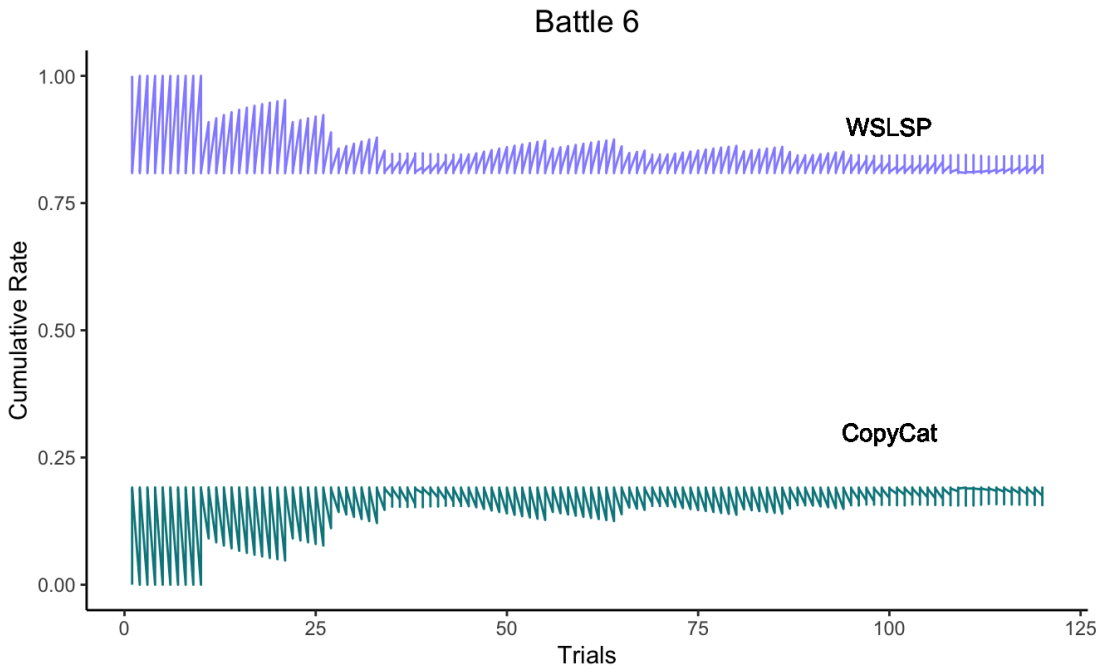


Appendix figure 10: Cumulative rates of CopyCat and WSLSP agents across 120 trials in Battle 5. CopyCat mean rate (0.814) with s.d. (0.02) and WSLSP agent mean rate (0.185) with s.d. (0.02)

Battle 6: CopyCat (mis-matcher) ‘v’ WSLSP (matcher)



Appendix figure 11: Distribution of WSLSP’s (matcher) wins and losses against CopyCat (mismatcher) agents in Battle 6



Appendix figure 12: Cumulative rates of CopyCat and WSLSP agents across 120 trials in Battle 6. WSLSP mean rate (0.816) with s.d. (0.01) and CopyCat agent mean rate (0.183) with s.d. (0.01).