



**MSc in Computer Science 2020-21**

**Project Dissertation**

**Project Dissertation title: Visualising Simulated Liquid Democracy and Measuring Performance Under Changing Agent Parameters**

**Term and year of submission: Trinity Term 2021**

**Candidate Number: 1049145**

## **Abstract**

More democratic than Representative voting, solving the knowledge issues of Direct voting, on the surface Liquid Democracy is a voting mechanism that appears to be an obvious tool to adopt. However, adoption has been scarce, and in some cases where it has been used it has had negative results.

This paper goes through the proposal and creation of a Liquid Democracy simulation program. The program visualises the voting mechanism to make it easy to understand. Also, it produces data and discussion that makes the benefits of the mechanism clear, while also showing how issues with it can arise and how to avoid them. The paper also serves as a summary of recent literature on Liquid Democracy. Some of the work from this literature is included in the created program. Future work and the previous work not included can be added to the program as extensions as it has been made open source.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Domain Description . . . . .	7
1.2	Problem and Contribution . . . . .	9
<b>2</b>	<b>Literature Review</b>	<b>13</b>
2.1	Existing Literature . . . . .	14
2.1.1	Power . . . . .	14
2.1.2	Delegation Mechanisms . . . . .	18
2.1.3	General Studies: Representativeness and Flexibility . . . . .	23
2.2	Technology Survey . . . . .	27
2.2.1	LiquidFeedback . . . . .	27
2.2.2	Google Votes . . . . .	30
2.2.3	Simulation of democracy.space . . . . .	32
2.3	Summary . . . . .	36
<b>3</b>	<b>Liquid Democracy Model Design</b>	<b>37</b>
3.1	Voter Agents . . . . .	37

<i>CONTENTS</i>	3
3.1.1 Delegates and Gurus . . . . .	38
3.1.2 Competence (Actual and Perceived) . . . . .	39
3.1.3 Power (and Visualised Weight) . . . . .	43
3.1.4 Preferential Attachment and Competence Importance . . . . .	44
3.2 Liquid Democracy Environment . . . . .	46
3.2.1 The Visualised LD Network . . . . .	46
3.2.2 Local Networks . . . . .	47
3.2.3 Delegations . . . . .	49
3.2.4 Voting and Results . . . . .	51
3.3 System Processes . . . . .	54
3.3.1 One-Shot Votes with Random Agents . . . . .	54
3.3.2 Continuous Votes with Learning Agents . . . . .	57
3.3.3 Visualised Simulations: Auto Run vs Analysed Run . . . . .	61
3.4 Systems UIs . . . . .	62
3.4.1 Setup Menu . . . . .	63
3.4.2 Agent Information UI . . . . .	64
3.4.3 End Results Menu . . . . .	65
<b>4 Liquid Democracy System Implementation</b>	<b>67</b>
4.1 Visualised Liquid Democracy . . . . .	69
4.1.1 Environment . . . . .	69
4.1.2 Agents . . . . .	71
4.1.3 Delegations . . . . .	74
4.1.4 The Voting . . . . .	78

4.1.5	Animations . . . . .	80
4.1.6	Agent Information UI . . . . .	82
4.2	Console Simulation . . . . .	83
4.3	Simulation Processes . . . . .	84
4.3.1	Creating Voter Agents . . . . .	85
4.3.2	Guru or Delegator . . . . .	87
4.3.3	Delegation Decision . . . . .	88
4.3.4	Voting and Ending the Round . . . . .	91
4.3.5	Learning Agents: Assessing Previous Vote and Delegations . . .	92
4.4	Simulation UI and Statistics . . . . .	97
4.4.1	Menu Screen: Setting Variable Values . . . . .	98
4.4.2	Statistics Screen and Manager . . . . .	99
4.4.3	Exporting Statistics to a CSV . . . . .	100
<b>5</b>	<b>Experimental Design</b>	<b>102</b>
5.1	Aims and Scope . . . . .	103
5.2	Dependent Variables . . . . .	106
5.3	System Study Setup . . . . .	107
5.4	One Shot Studies . . . . .	108
5.4.1	Independent Variables: Preferential Attachment and Competence Importance . . . . .	109
5.4.2	Independent Variable: Network Size . . . . .	110
5.4.3	Independent Variable: alpha . . . . .	111
5.5	Learning Agents Studies . . . . .	112

5.5.1	Direct Voting vs Liquid Democracy . . . . .	113
5.5.2	Independent Variable: alpha . . . . .	114
<b>6</b>	<b>System Study Results and Discussion</b>	<b>115</b>
6.1	One Shot: Effect of Changing Agent Values . . . . .	116
6.1.1	Results of Preferential Attachment Experiment . . . . .	116
6.1.2	Results of Competence Importance Experiment . . . . .	118
6.1.3	Discussion . . . . .	119
6.2	One Shot: Effect of Changing Local Network Sizes . . . . .	122
6.2.1	Results of Network Size Experiment on Med Competence Com- munities . . . . .	122
6.2.2	Results of Network Size Experiment on Low Competence Com- munities . . . . .	125
6.2.3	Discussion . . . . .	127
6.3	One Shot: Effect of Changing Alpha Value for Network Sizes Experiment	130
6.3.1	Results of Network Size Experiment for 3 More Alpha Values .	130
6.3.2	Discussion . . . . .	133
6.4	Learning Agents: Direct Voting vs Liquid Democracy . . . . .	136
6.4.1	Results of Direct Voting vs Liquid Democracy . . . . .	136
6.4.2	Discussion . . . . .	140
6.5	Learning Agents: Liquid Democracy with Different Alpha Values . . . .	142
6.5.1	Results of Learning Agents using Liquid Democracy with Differ- ent Alpha Values . . . . .	142
6.5.2	Discussion . . . . .	143

<i>CONTENTS</i>	6
<b>7 Conclusion</b>	<b>146</b>
7.1 Contribution . . . . .	146
7.2 Future Work . . . . .	147
7.3 Overview and Reflection . . . . .	148
<b>Appendices</b>	<b>153</b>
<b>Appendix A Open Source GitHub and Advance Settings Screen for Ex-</b>	
<b>tension</b>	<b>154</b>
<b>Appendix B Video Demo of Liquid Democracy Simulation Program</b>	<b>155</b>

# Chapter 1

## Introduction

### 1.1 Domain Description

Liquid Democracy is a decision making mechanism that can be used for collecting votes to make a decision within a community, where a community can be anything ranging from a single organisation to a whole country making political decisions.

Also known as transitive proxy voting (Harding, 2021), in a liquid democracy voters may choose to be Delegators or Gurus. Gurus are voters who vote directly on an issue, whereas Delegators delegate their vote to a proxy, who may then use their own vote and the votes they have been delegated to vote on an issue as a Guru, or may also delegate their own vote and the votes they have been delegated to another voter, hence the term transitive. Gurus vote with a power of one plus the number of votes they have been delegated either directly or through transitive delegations.



Liquid Democracy is often seen as a best of both worlds compromise between direct democracy and representative democracy (Kahng, Mackenzie and Procaccia, 2021). In direct democracy each voter in a community votes directly on each issue, giving complete flexibility of votes. A major issue raised with direct democracy is that you cannot reasonably expect every voter to be well educated on every issue area (Green-Armytage, 2014), when thinking about the issues a country may vote on if voters had to learn about all possible vote topics they would have little time for anything else. So while direct democracy gives the flexibility that some see as true democracy, forcing voters to vote directly if they want to vote could end up with undesirable decisions for the community due to lack of topic knowledge.

Representative democracy is where the majority of members within a community vote for a representative, and the elected representatives then vote on each of the individual issues, an obvious example being the UK government where the population elect local MPs, and those MPs then vote on the individual issues for the UK. Some however, have deemed representative democracy not democratic enough (Green-Armytage, 2014) due to the lack of flexibility of being stuck with the same representative for a given amount of time or votes. Indeed, it is unlikely the representative a voter votes for will agree with the voter on every single issue, meaning they may vote against the voters interest, and the voter has to wait for the next time representatives are elected to again have their say.

As stated, with the liquid democracy mechanism voters have a choice, they can vote themselves on an issue or delegate to a Guru. The important thing to note is they can vote differently every time there is a new vote, voting directly on issues they really care about and understand, and delegating on issues they are less sure about to more “issue competent” voters (Blum and Zuber, 2015). The Guru they delegate to can be someone they know and trust has more knowledge about the issue, they can have different Gurus for different issues, and can stop delegating to a Guru who votes against their interests straight away. The ability to freely choose to vote directly means the system has the flexibility of direct democracy, and the ability to have a Guru as your representative deals with the knowledge issue, showing why liquid democracy is a desirable mechanism.

## 1.2 Problem and Contribution

The concept of Liquid Democracy has been around since about the year 2000 (Mendoza, 2015), yet there has not been any major successful permanent uptakes of the mechanism as of yet. There has been some experimentation with the idea. For example Google used their liquid democracy system “Google Votes” to make office decisions such as what food to have in the cafeteria (Hardt and Lopes, 2015), allowing users (the employees) to delegate their votes through the now defunct social media site Google+.

Maybe more notably, there have been some real world experimentations with liquid democracy implementations in the political domain through some local parties, such

as the Pirate Party in Germany, Demoex in Sweden, the Net Party in Argentina, to name a few (Kotsialou and Riley, 2020). These real world implementations did however expose some flaws in the liquid democracy approach (Gölz et al., 2018), a major one being the creation of “Super Voters” who amass a huge number of votes and basically control the vote whereas many other voters get no delegations. These real world political domain experiments show the mechanism needs to be well understood in order to be implemented in a way that works for a given community without issues arising. So, a model of liquid democracy that can help understand how it works when you change different factors would be beneficial.

A number of Liquid Democracy models have been coded for experimental research. These are typically console based programs without a UI that allow the experimenter to study the effect of changing different variables within a Liquid Democracy system, such as how much voters desire power (Zhang and Grossi, 2021), or the effect of trying different delegation mechanisms (Kotsialou and Riley, 2020). While these single purpose models can be helpful in showing how liquid democracy performs with these different factors changed and therefore help decide how to implement a liquid democracy system, for example the delegation mechanism experiment found the “Super Voter” issue could be counteracted by using a breadth-first delegation rule (Kotsialou and Riley, 2020), they only focus on a single part of the system and they do not help understand how liquid democracy actually works.

This paper proposes that what is needed is a Liquid Democracy simulation software

that actually visualises how the mechanism works through the use of a UI. This unlike the models mentioned above will then be actually understandable to laypersons who will both not need to be an expert in liquid democracy or have to understand any programming language, as the simulated visualisations will show the system running down to an individual agent representing each individual voter, clearly showing if they are a Guru or a Delegator, the delegations the Delegator agents make, and how they vote.

The proposed simulation program will also have variables that can be edited by the user, such as the number of voters in the community, how much they understand the vote topics, and more. These variables allow them to see how different possible scenarios could affect the quality of the votes in a given community, and to see what values or options to set factors that can be controlled for the best possible implementation. So just like the models of liquid democracy it will have the potential use as a testing ground for the conceptual and empirical claims put forward by supporters of liquid democracy (Harding, 2021), hopefully having other benefits such as balancing the practicality and democracy in a real world implementation and increasing voter competence and turnout with using a liquid democracy system.

So in summary, the proposed visualisation program should be able to be used to prove theorems about liquid democracy as well, and should ideally be able to be extended to include other variables to allow for more experiments, giving people who want to understand the mechanism the ability to use this one program to look at all

possible factors, instead of having to research the outcomes of many individual models that need specialist knowledge to understand as they have no visualisation.

# Chapter 2

## Literature Review

The original concept of Liquid Democracy (LD) was described by John Washington Donoso (Mendoza, 2015); the exact date is unknown as it was first mentioned in forum pages that are now inaccessible, but it was sometime around the year 2000. It was originally a system that could recommend better quality answers to the questions an online user might have, and was not meant to be a decision making tool. While the original concept did propose the idea of dynamic vote delegation, John himself stated it wasn't for decision making for use in traditional government, and he definitely didn't see it as a "quick fix" replacement for current voting methods. Mendoza stresses that the change from quality answers to decision making has been a non-trivial one.

This section will be a literature review on LD, acting as a summary of the key research and implementations that have arisen since 2000, which is arguably overdue considering the growing amount of papers and interest in the subject area. The section

covers some of the papers written on LD briefly looking at the models used but mainly focusing on the independent variables assessed in the research and the findings, while also looking at the created technology through any key programs and implementations, and briefly covering the outcomes of real world political usage of LD.

## 2.1 Existing Literature

LD is a relatively new field so there is not a huge body of work available, however this subsection will cover the key work that has been done so far, splitting the work into categories depending on the purpose of the research.

### 2.1.1 Power

In the domain of LD, power refers to how much weight an agent could have behind their vote (e.g. how many delegations they have currently received). A potential issue with transitive delegations is that they could lead to disproportionate accrual of power (Zhang and Grossi, 2021), which in turn would harm the legitimacy of resulting votes. Zhang and Grossi created a model to account for power-seeking behavior by agents, where agents do want to vote truthfully but also consider how much power they will retain in the system. So agents in their system choose their delegations by aiming to optimise the trade-off between pursuing high accuracy and seeking more power (having higher influence).

The power-seeking model by Zhang and Grossi provided a generalisation of the

power index known as Banzhaf index; to account for delegations in voting with quota rules, they called it the “delegative Banzhaf index”. Their power index measured the power of both voters and Delegators, so it did not assume Delegators have no power once they have delegated. In their model direct delegations were worth more than transitive delegations, so a Guru obtaining  $N$  direct delegations is more “powerful” than a Guru obtaining  $N$  delegations from a Delegator who has received  $N - 1$  direct delegations. Additionally their model had a binary voting setting (2 options), and delegations were constrained by an underlying social network (as a representation of the relationship of which agents were aware of each other) which was represented by a directed graph.

The experiments found that groups where agents are more power-greedy appeared to achieve more equal distributions of power (Zhang and Grossi, 2021). In the experiments as power became more important to agents they became more reluctant to delegate initially, also agents were found to prefer shorter chains to their Gurus when power was more important, these shorter chains resulting in more equal power distributions. They also showed that limiting the level of connectivity of the underlying network is beneficial in limiting inequalities in the power distribution.

In building a simulation of LD the effects on the results by these different distributions of power which resulted from the different connectivity which in turn resulted from different weights on power greed could be interesting to explore. The actual “delegative Banzhaf index” model of power however may not be the best representation of power within LD and would need to be explored further if including this functionality



in the system, as in the results of the experiments it was noted that the average power increases and decreases (Zhang and Grossi, 2021) (due to the model assumptions on Delegators power and “influence”), but it may be more reasonable to assume there is always a constant power in the system, as the number of votes does not change.

Power is actually at the centre of one of the biggest weaknesses of LD that was briefly mentioned in the introduction. The issue of “super voters” (Gölz et al., 2018), that was exposed through real world use, where super voters gained massive weight (power) behind their votes where most voters got little to no delegations; leading to one specific case in the German Pirate Party who experimented with LD having a linguistics professor at the University of Bamberg who amassed so much weight that his “vote was like a decree” (Kahng, Mackenzie and Procaccia, 2021).

Because of this issue of agents in a LD system acquiring too much power, there has been research on how to implement a LD system that minimises the maximum power in a LD vote (Gölz et al., 2018). Gölz et al built a model that aimed to both solve the algorithmic problem of selecting delegations to minimise the maximum voting weight of any agent in the system, while also showing that allowing multiple delegation options for voters significantly reduced the maximum weight within a community of voters.

The goal of the research by Gölz et al did not look at the result of the voting, they did not care about getting a particular outcome, they just wanted to change the process of LD to reduce the effect of super voters and allow more voices in the system

to be heard. In the voting process if agents chose to delegate they were able to specify multiple potential delegations. The aim of the system was then to choose a delegation for every Delegator in such a way that the maximum weight of any voter is minimized. It was compared to the problem of congestion minimisation for confluent flow, which is closely resembles, throughout the research. The issue of delegation cycles was raised, where voters delegate to each other so there is no Guru to assign the voting power to; this model leaves these votes unused. The measure of success for the experiments of their system was of course its effect on the maximum weight in a vote.

Resulting experiments showed that allowing multiple delegation options significantly reduces the maximum weight (Gölz et al., 2018), as well as finding that it is computationally feasible to resolve delegations (choosing which of each voter multiple potential delegates to actually delegate to in order to minimise the maximum weight of each voter) in a way that is close to optimal. The returns of increasing the number of allowed potential delegations quickly diminish, doubling from 1 to 2 showing the most significant reduction in maximum weight and very little reduction for each further increment, which is common to many incarnations of the power of choice. The system had numerous different models for picking delegations which won't be fully covered here, but a generalised greedy algorithm was created that was very fast, especially compared to the other models, but still performed comparably as well as them, which lead Gölz et al to conclude that the results suggest that it would be possible to resolve delegations almost optimally even at a national scale.

The system Gözl et al built and experimented on selected one Guru per Delegator, an alternative that could be explored would be to split the power of the Delegator's vote between all their specified Gurus (Gözl et al., 2018), as it would be interesting to see the effect such a system would have on the weight distribution. It is important to note that in their system they also allowed voters to change their vote if they disagreed with the choice made by their Guru which could only further reduce the maximum weight in their experiments. Again, the research here just focused on the maximum power; it would be interesting to see how the results of a vote changed when allowing multiple potential delegates to be able to see the relation between maximum powers and the results.

### 2.1.2 Delegation Mechanisms

In the work on limiting power by Gözl et al they change how the agents (or users) in the system can delegate their votes, allowing multiple potential delegations instead of just one. The idea of allowing different ways of delegating has actually been explored in a few different ways, both looking at how a Delegator may delegate, and also how the delegation is assigned in cases when multiple delegations are allowed, these are called "Delegation Mechanisms".

There are a number of different voting mechanisms that can be used by voters besides just picking one delegate and ignoring the rest, for example approval voting and ranked voting (Hardt and Lopes, 2015), where in approval voting a Delegator may approve of multiple delegates, where either one is chosen as the Guru or the power is

split, and in ranked voting a Delegator can select multiple delegation options and rank them in their preferred order, with the ranking typically ending with themselves or abstaining. Then, in cases when you have multiple delegates in a ranked or preferred order there are multiple mechanisms for selecting a Guru from the transitive chain of delegations, such as breadth-first and depth-first delegation (Kotsialou and Riley, 2020).

As previously stated, the success of the idea of LD is the best of both worlds compromise between direct and representative democracy, with the advantage against direct democracy being that not every person needs to learn about every issue, they can just delegate their vote. But an important question is if there are any delegation mechanisms that are guaranteed to yield more accurate decisions than direct voting (Kahng, Mackenzie and Procaccia, 2021), where “more accurate” means getting more correct results from votes that lead to a good or desired outcome for a given community. Kahng et al did research looking into whether LD with certain delegation mechanisms performed better than direct voting, with a focus on the same issue as Gözl et al, the issue of individual voters gaining too much power, but unlike Gözl et al their concern was on the results of the votes within the system. The research looked at two types of delegation mechanisms, local and non-local mechanisms.

The model built by Kahng et al represented the problem using a directed labeled graph, where edges in the graph represent a “social network” with an edge from a voter A to a voter B means A knows of B. Their model assumed that the voters vote on a binary issue, which meant there were two options, with one being the correct option

that leads to the best outcome for the community, and the other being the incorrect alternative. Each voter had a competence level representing their expertise in the issue and therefore how likely they were to choose the correct option in a vote, and the model assumed voters would only delegate their vote to higher competence voters, specifically voters with a competence at least a certain approval value higher than their own, which meant there could be no cycles in the delegations.

Local mechanisms are said to capture the spirit of liquid democracy (Kahng, Mackenzie and Procaccia, 2021), as they make delegation decisions based only on the neighborhood of each voter, the voter makes their delegation decision from their own viewpoint, without central coordination or knowledge of global properties about the delegation graph. An example given by Kahng et al of a local voting mechanism is where each voter delegates their vote, if they have any approved neighbor, to one at random, other examples are also based on local knowledge (e.g. approval). Non-local mechanisms require a centralized authority, i.e. an algorithm that coordinates delegations. A couple of the examples given are one, where each voter delegates their vote to their most competent approved neighbor, and two, where each voter delegates their vote to an approved neighbor who does not, in turn, delegate their vote to anyone else.

The success of the delegations mechanisms was measured by Kahng et al as the “gain”, with the gain being the increase in the number of correct results. For local mechanisms it was found that although they could result in a positive gain, they also introduced significant correlation between the votes, leading to the familiar problem

of high weight “super voters”, and when these high weight voters are incorrect the weighted majority vote is incorrect, whereas direct voting is very likely to lead to a correct decision, meaning a negative gain, so local mechanisms could not guarantee more accurate results. Non-local mechanisms however circumvent the issue of heavy weight voters, as you have a central authority that can look over all delegations, allowing a system that was called “GreedyCap” to imposing a cap on the number of votes that can be delegated to any particular voter, thereby avoiding excessive correlation. So Non-local mechanisms always had a positive gain, outperforming direct voting.

The work by Kahng et al had a number of assumptions, such as binary right and wrong answers to issues, and voters only delegating to higher competence voters, that seem realistic and could be helpful in building a simulation of LD. However, the idea of non-local mechanisms seems to lose the spirit of LD (Kahng, Mackenzie and Procaccia, 2021), and it could be argued some of the non-local mechanisms based on competence could be worked into a local mechanism simulating the perceived competence of a potential Guru communicated by word of mouth between voters in a local network.

How to handle personal rankings over delegations to voters is one of the main ongoing issues in LD in general, not just for getting more correct results, but for many things, such as incentivising participation within a LD system (Kotsialou and Riley, 2020). Kotsialou and Riley investigate the effect of two different delegation mechanisms on participation, one being depth-first delegation which they identify issues with, and breadth-first delegation which was newly proposed by them to guarantee better partici-

pation for casting voters (Gurus) in the system. Depth-first and breadth-first delegation can be understood at a simplified level as seeing the transitive delegations as a graph, and how each mechanism would pick a Guru from said graph by having the Delegator as a starting node and locating the closest available Guru, see their paper for more detail.

The participation property holds if a voter, by joining an electorate, is at least as satisfied as before joining (Kotsialou and Riley, 2020). This property had to be split into two separate properties by Kotsialou and Riley due to the delegation option. The first property was “cast participation”, where a voting rule satisfies this property when every voter weakly prefers joining any possible voting electorate compared to abstaining, regardless of who is in the delegating electorate. The second was the “Guru participation” property which was the focus of the paper, where for any casting and delegating electorates a voting rule satisfies this property when any voter weakly benefits from receiving additional voting rights of any voter.

Kotsialou and Riley showed that the different delegation rules can have different properties, specifically looking at Guru participation. Their study produced a theorem for and proved that depth-first delegation does not guarantee Guru participation when the delegation graph contains cycles, also highlighting that if there are no cycles then Guru participation is guaranteed. They also prove a theorem that states that the proposed breadth-first delegation mechanism does always guarantee Guru participation, meaning they proved that it has the desirable property that every Guru weakly prefers

receiving delegating voting rights.

This shows the importance of delegation mechanisms if you allow multiple potential delegates, as in this model only one of the mechanisms always satisfies the desired property of Gurus preferring to receive additional voting rights (Kotsialou and Riley, 2020), and you want Gurus to feel that it is worth being a Guru. Kotsialou and Riley also acknowledge the issue of representation by a small set of high weight Gurus, showing how big this issue is with so many referring to it, suggesting it may be possible for breadth-first delegation to resolve the issue as it favours keeping delegated voting rights close to their origin.

### **2.1.3 General Studies: Representativeness and Flexibility**

In terms of democracy systems Liquid Democracy is still very much a new and growing field, so there have been many different areas of LD that have been covered but not in big enough quantities to categorise into a single group like above. This subsection will cover some of the other latest work in the field of LD.

A key part of LD is the option to be represented through proxy voting, in the case of a voter choosing to delegate and not vote directly it is desirable for there to be a proxy in the system they can delegate to that represents them and their values well (Anshelevich et al., 2020). An algorithmic approach to the issue of representativeness in an LD system was taken by Anshelevich et al, with their goal being to find how much time it takes to find a set of proxies that provided a certain level of representation for





Figure 2.1: Representation of voters, proxies, and candidates on a line, where the closer together they are the more they relate to each other, provided by Lirong Xia in a presentation of their work.

Delegators in the system.

In the model of LD made by Anshelevich et al, voters and proxies are placed on a line, the closer things are positioned on the line the more their values co-align. An example of this can be seen in Figure 2.1, with two voter “v” and “w”, and two proxies represented by blue triangles, also on the line are 3 red square representing candidates as in their model it is assumed voters vote for candidates, but in LD you should not need candidates, however you can just as easily picture these candidates as policies so the assumption does not restrict the model. In Figure 2.1 voter “v” is well represented as their closest proxy is p1 who would in turn vote for C1 which is the closest and most relatable to “v”, but “w” is not well represented as the closest candidate is C2, however the most relatable proxy is p2 who would vote for C3.

The goal of Anshelevich et al was to find a set of proxies that is  $\theta$ –representative, so for every voter on the line its favorite candidate is within a distance  $\theta$  of the favorite candidate of its closest proxy. Proxies are created from scratch which they note may seem unnatural, however argue it is reasonable to assume that human voters would find

a proxy more relatable than the actual candidate.

It was found that it was possible to compute a  $\theta$ -representative arrangement with the smallest number of proxies in polynomial time (Anshelevich et al., 2020), and also, while the detail won't be covered here, efficient algorithms for computing optimal sets of  $\theta$ -representative proxies were given, as well as proved upper and lower bounds on the number of proxies needed to achieve this property. It is noted that the proxy arrangements do not depend on the voter locations, and therefore will remain representative for any set of voters.

One main advantage of the LD framework, its flexibility, did lead to concern about its stability (Escoffier, Gilbert and Pass-Lanneau, 2020), with it being shown that equilibrium of LD's delegation process may not exist, and that the existence of such an equilibrium is even NP-hard to decide. Escoffier et al looked at various natural structures of preference over Gurus to see if they guaranteed the existence of an equilibrium; these different preference profiles (how voters order which Gurus they want to delegate to) can be seen as quite similar to delegation mechanisms, but focus instead on how agents in the system approve of each other instead of how those preference orders are used.

The first type of preference profile Escoffier et al studied was 'single peaked', where agents are ordered on a line and they prefer Gurus that are closer to them on the line, much like the model of representation used by Anshelevich et al. The next was "sym-

metrical preferences”, where all pairs of voters always accept each other as Guru, or reject each other, so if agent A only accepts agent B as a Guru if B accepts A and vice versa. The final profile was “distance based” where voters are embedded in a metric space and accept the voters that are close to them in this space as possible Gurus, so like symmetrical preferences but with two dimensions.

It was found that even though the existence of an equilibrium of the LD process is NP-hard to decide when preferences are unrestricted, the introduction of restricted types of preference on the agent representing voters can guarantee the existence of an equilibrium (Escoffier, Gilbert and Pass-Lanneau, 2020), with all three profiles in the study mentioned being able to guarantee an equilibrium.

Sticking with the topic of flexibility, there has been an attempt to increase the flexibility of LD further by introducing the concept of “pairwise liquid democracy” (Brill and Talmon, 2018), which considered ordinal elections, where each voter needs to order a set of alternatives by how they rank them. Brill and Talmon state that this flexibility is useful for voters who do care about the outcome of an election, but do not feel competent enough to compare certain pairs of alternatives. But also acknowledge the potential price of the extra flexibility, being the possibility of intransitive preference orders resulting from refining a partial order by delegating certain pairwise comparisons to different delegates.

Brill and Talmon took their generalized model of liquid democracy where voters can

delegate pairwise comparisons, and explored approaches to deal with the possibility of intransitive preferences. Approaches they explored included restricting allowed delegations, looking at trying to circumvent intransitive preferences by ignoring delegations, and the possibility of consolidating intransitive preferences into weak rankings. However, this model of LD, while having advantages of added flexibility, seems possibly too complicated for the proposed visualised simulation of LD by this paper, as it is meant to help with understanding LD and these ranked alternatives add a new layer of complexity. While dealing with intransitivity in this model was important, the simulation created in this paper uses only single delegations.

## 2.2 Technology Survey

### 2.2.1 LiquidFeedback

Liquid Democracy has in the past been confused with “LiquidFeedback” (Mendoza, 2015), both employing the term liquid in their names, they should not be confused as LD is the concept where LiquidFeedback is just LD software (Figure 2.2), in fact it is the software that was employed by the German Pirate Party. Developed by the “Public Software Group e. V.”, the idea of the LiquidFeedback computer software is to “empower organizations to make democratic decisions independent of physical assemblies while also giving every member of the organization an equal opportunity to participate in the democratic process” (Behrens et al., 2014). LiquidFeedback is platform independent as it can be accessed through a web browser, so nothing needs to be installed.

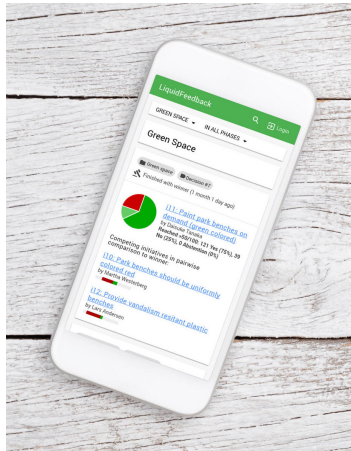


Figure 2.2: LiquidFeedback UI on a smart phone from the LiquidFeedback website, displaying a LD vote on painting parking benches that has finished with the proposal to paint them green on demand winning.

The book on the principles of LiquidFeedback state it was designed to make binding decisions of any topic for political parties and organizations of any size (Behrens et al., 2014), combining concepts of a collectively moderated, self-organized discussion process (constructive feedback) and Liquid Democracy (proxy voting). The system covers the whole process from the first draft of a proposal to the final decision. This allows all users not only to participate in voting but also be involved in the development of ideas. From the point an idea is proposed the designers of the system only want constructive feedback within the idea's discussion, in the case users are against the idea they should start or support a counter proposal, or simply just vote no against the original proposal, this is to make sure everyone within the discussion of an idea are working towards the same goal.

The LD voting system in LiquidFeedback has users selects a single Guru for different

topics (Kotsialou and Riley, 2020), but also allows voters to express preferences over options that are stated to be counted “using a sophisticated system” that the designers based on recent social choice theory research (Behrens et al., 2014), with the reason being to not force users into making undesired compromises or trying to vote by majorities or chances. Behrens et al state the even if the system is only used to collect results, that aren’t official or binding, it can still be used to indicate to representatives in charge the desire of the people within the community and help them with their decision making. This would allow a community to get used to the system without enforcing the decisions made by using it. Both traceability (seeing what is done with votes and how decisions are achieved) and being verifiable are stated in the LiquidFeedback book while talking about the design of the system as being important for a decision making process to be trustworthy.

The success of LiquidFeedback is still up for debate, in terms of adoption of LD it has done very well, with the German Pirate Party, which was one of the first organizations to employ the LiquidFeedback software, raising the popularity of LD (Brill and Talmon, 2018), leading to other parties such as the Spanish party Partido de Internet and the Argentinian “Net Party” promising to user similar tools. The actual usage within a real political party was however less successful, with the structural flaws in the system leading to its demise and the system being taken offline for the party (Mendoza, 2015). The usage of the tool lead to what was called Liquid Wars: “the mother of all battles among factions”, and also having one super-voter (a voter who amassed a huge majority of delegations) who had a power that made their vote “like a decree,” even though he

held no office in the party (Gölz et al., 2018). So while the online platform has helped with adoption and awareness of LD, the LiquidFeedback system itself was floored and needed improvement before real world usage.

### 2.2.2 Google Votes

As briefly mentioned in the Introduction, Google created their own LD system built on their own internal social media network Google+, and was implemented to perform an experiment of using LD in a corporate environment to make decisions (Hardt and Lopes, 2015). The study on Google Votes notes a social network’s user relationship graph bears similarity to a LD delegation graph, and allows Gurus to be people well-known to those who delegate, instead of representatives who are likely not well-known.

The Google votes system followed ”The Golden Rule of Liquid Democracy”, which means that if a user delegates their vote they can see what is done with it (Hardt and Lopes, 2015) (much like the importance of traceability stated in the LiquidFeedback book). Google+ social media posts were used as the basis for vote and issue discussions, with discovery of relevant issues to vote on and vote recommendations from other users in their social network. The system allowed different voting types, namely: yes/no, plurality, approval voting, score voting, and ranked voting.

The Google Votes system had a focus on educating its users about LD allowing them to take full advantage of the mechanism (Hardt and Lopes, 2015), typically using familiar concepts from representative democracy to describe aspects of LD. In order to

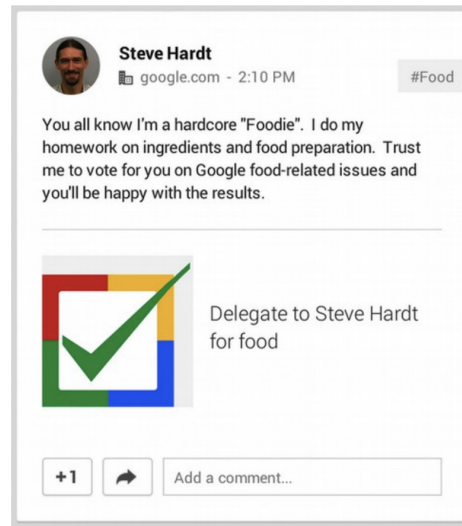


Figure 2.3: A Google Votes delegation advertisement posted by a Google Votes user, with the user explaining their expertise on the topic of food, informing other Google Votes users why to pick them as their Guru for food related issues.

push the creation of delegations, the Google Vote system allowed delegation advertisements (Figure 2.3) where users could tell other users why they should delegate their vote to them. They also included teaser messages telling voters they could increase the power of their vote to try and intrigue them to use the delegation features, as well as displaying the “estimated power” of the voter when they vote to make them aware it could be increased.

Delegations were done by category in Google Votes, meaning users set different delegates for different issue topics, with a delegation just being a directed link from one user to another user, annotated with the issue category. User’s had a voting page where they could add and remove delegations, but it is important to note if they did not like the vote the user they delegated to cast they could override it and vote for themselves, this



is why in the system displayed users “estimated power” and not just “power”, as they may not retain the full power of their vote if some of their delegates chose to override it.

The Google Votes experiment took place from March 2012 to February 2015, with 15,000 users casting over 87,000 votes on 370 issues (Hardt and Lopes, 2015). The “food” category of issues had both the most number of issues raised and number of votes cast, being 150 and 73,000 respectively, which was to be expected due to Google having a strong food culture with free snacks and meals at on-campus cafes. 3.6% of the total vote count were delegated votes, which while a small amount was seen as significant due to the concept of delegated voting being new to most.

Hardt and Lopes conclude that the study both provided real-world Google Votes usage, as well as the food voting project setting and meeting goals to increase participation in food-related decisions, therefore improving the efficiency of the Google Food team. They go on to state their belief that direct and representative democracy systems have not lived up to the potential of democracy and that practical liquid democracy systems combining the best of direct and representative democracy systems are now possible.

### **2.2.3 Simulation of democracy.space**

In 2019 David Ernst alongside Cy Ray did a presentation for the Youtube Channel “Simulation”, with the topic being Liquid Democracy (Simulation, Ernst and Ray, 2019). Their presentation included a live working simulation of a LD system, with

the purpose of advertising the real LD system at the time known as “liquid.us”, which has since been rebranded to “democracy.space”. The democracy.space online tool is an LD system that is advertised as “making a new kind of democracy possible” (democracy.space, 2021), allowing users to vote on policies with the key features of LD, being able to vote directly when they want and choose proxies to represent them the rest of the time.

Ernst, one of the presenters of the simulation and system, has talked at great length about LD before, with many online articles on the topic. He sees the key to LD being “networks of trust” (Ernst, 2016), talking about being able to delegate to friends, family, your teachers and professors, which allows LD it to flourish at scale. He talks about the issue with “winner takes all” elections being they entrench a two party system, and the great thing about LD is everyone keeps the power they are delegated in making decisions, alongside the fact you can only run elections every few years, where in LD you can change you delegations instantly from your phone, allowing for true accountability.

In the Simulation presentation LD is presented following the idea that it is the best of both worlds of direct and representative democracy, but alongside just giving information about LD systems they provide simulations to show how it works (Simulation, Ernst and Ray, 2019), showing a 2D visualisation of LD where each person starts out on equal footing (Figure 2.4 (right)), as well as showing electoral democracy for comparison, where you have a politician with all the voting power (Figure 2.4 (left)).

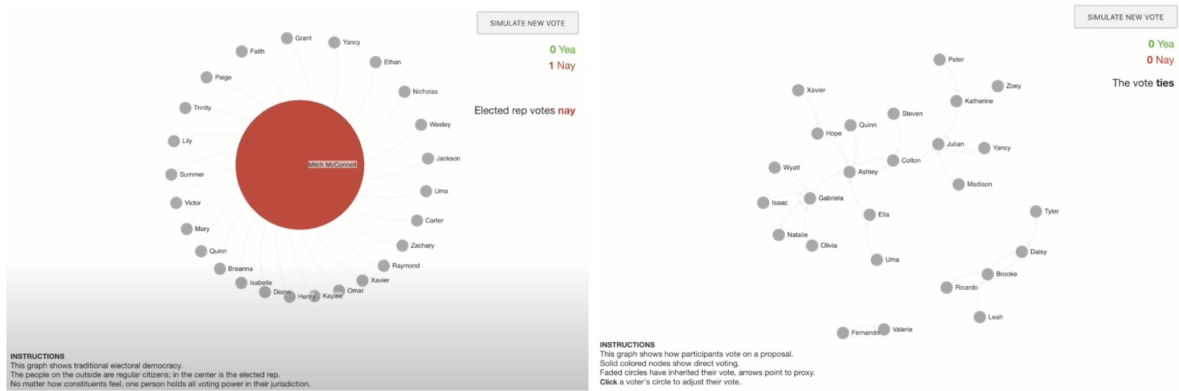


Figure 2.4: Left: Simulation of Electoral Democracy with one agent having all the power, Right: Simulation of Liquid Democracy in start state with all agents on equal footing.

The LD simulation (Figure 2.5) they created visualises voters as nodes in a 2D graph with delegations being directed edges between them (Simulation, Ernst and Ray, 2019), showing the flow of power as a chain of delegation edges. Those who inherit their vote through delegation have a faded colour when they vote, those who directly vote have a solid colour. Each voter can vote directly even if they have delegations, they only inherit the vote of who they delegated to if they choose not to vote themselves, hence why you can see delegations such as from Julian to Colton and Leah to Brooke in Figure 2.5 that do not take effect.

The current result of the vote shown in the simulation (Figure 2.5) shows “Nay” winning, however this implementation of LD does allow voters to override if they don’t like their delegate’s vote (Simulation, Ernst and Ray, 2019). At the 30:30 timestamp in the demo video you can see Ashley override her vote, shifting the result for “Yay” to win, as she was the source of the majority of Colton’s votes, who before the most

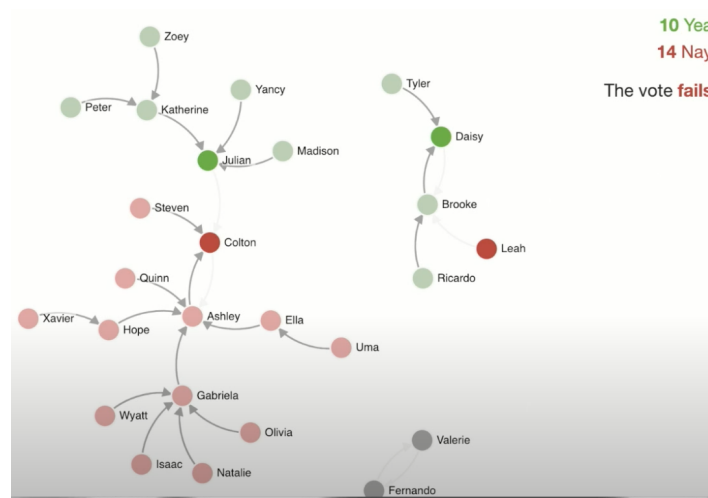


Figure 2.5: Simulation of Liquid Democracy shown on the "Simulation" Youtube channel, showing votes and delegations, with "Nay" currently in position to win the vote.

powerful voter, showing the importance of direct delegations compared to transitive, as with Ashley removing her delegation Colton goes from 13 votes to only 2.

After the demo Ernst emphasizes the idea that "you know you are representative" with LD (Simulation, Ernst and Ray, 2019), and goes on to talk about "Proof of Vote" system included in their LD implementation to avoid corruption of votes so the result can be trusted.

While the system they were demoing for has since evolved in democracy.space, the basic principles are still the same, also introducing scorecards that grade politicians (who don't follow LD rules) for how much they follow their constituents (democracy.space, 2021), while pushing for candidates that are pledged to vote as directed by their local liquid democracy. The democracy.space website shows a number of bills that

have been voted on (in U.S. Congress) using the system, again showing how the system works while also showing there has been some adoption, perhaps helped by the number of articles written by Ernst and co as well as them visualising how the system works to help people understand what they would be joining.

## 2.3 Summary

For a relatively new idea there has been a lot of interest in research into liquid democracy. Many different models have been created that show the benefits of the voting system, and others have shown how to overcome the issues. However there has been little real world up take, and where there has been uptake in political settings the issues, such as super votes, have prevailed. This does not help sell the idea of adopting liquid democracy.

Communities are thinking about using liquid democracy in real life should not need to find lots of individual papers and models to see the benefits it provides and the issues to avoid. That is why this paper aims to create simulation software that makes the system easier to understand while making clear how to get the benefits and avoid the issues. The idea is the software includes parts of some of these models, and extensions can be made in the future to include even more.

# Chapter 3

## Liquid Democracy Model Design

This section describes the designs of the Liquid Democracy (LD) program that this project aims to implement. There are two distinct parts of the design to be considered that will be described in parallel throughout. Firstly, the design of the underlying model of the LD system, referring to how it will be implemented by code, looking at how agents will work, what parameters they should have, how the delegation mechanism will work, and other system variables that could be included. The second are the literal designs of how parts of the LD will be visualised to the user in the visual simulation part of the programs, showing, for example, agents and their delegations to the user, aiming to make it clear how the underlying LD model works.

### 3.1 Voter Agents

Each voter in the LD system will be represented by an individual uniquely identifiable agent who acts within the created environment, the terms voter and agent will be used

interchangeably from this point. These agents are arguably the most important part in defining a LD voting system, in fact in designing any voting system, as they represent the real people who will be using said system if ever implemented in real life. These people's lives will be affected by the decisions made using the system and so deserve to be shown they can trust that they are being given true democracy and that the system truly works in achieving the best for them and their community.

### 3.1.1 Delegators and Gurus

In a Liquid Democracy each voter can choose to vote directly on an issue or give their vote to another voter, a proxy, to decide what to do with their vote for them. The voters who choose to vote directly are called “Gurus”, these Gurus may have only 1 vote towards an issue (just their own). or a number of votes including their own plus however they have been delegated either directly or transitively. Voters who give away their vote or “delegate” their vote to another voter are known as “Delegators”. They choose another voter they know in the system to receive their vote, their vote may be used by that voter directly if they are Guru or passed on further if that voter also chooses to be a Delegator, eventually reaching a Guru at the end of the transitive delegation chain who will represent the original Delegator as well as all the others in the chain for the vote on the given issue.

In the visualised simulation of the LD system there will be a clear distinction made in the representation of Delegators and Gurus. To start, each agent will be represented individually in the system with a humanoid figure (figure 3.1 (left)), designed similarly

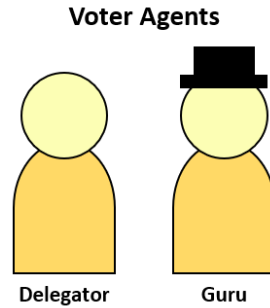


Figure 3.1: Design of the Voter Agents, Left: Delegator, Right: Guru

to a pawn in a game of chess, to make a clear relation between real people and the agents. Each agent will start the simulation looking the same, looking like the model on the left of figure 3.1. However, once they make the choice of Guru or Delegator the user of the system will be able to tell them apart, with Delegators keeping the original look and Gurus adopting a new look (figure 3.1 (right)) adding a hat on their head to represent they are officially voting on the current issue.

### 3.1.2 Competence (Actual and Perceived)

In real life different people know different amounts about different issues, so when it comes to voting on certain topics some people will be better informed than others and therefore more likely to make a good decision. This is a key point of representative democracy discussed earlier. You cannot expect everyone to learn about all the issues that may need to be voted on, so you have representatives whose job it is to know about these issues, and have voters vote for a representative who then votes on the issues directly. So in a LD where everyone can vote directly this is important to consider, each agent should have a set knowledge about issues, a “competence” when it comes



to making a right decision.

The type of voting this LD model will use is a binary voting system, this will be discussed more later on, but in short means there are two options that each voter can pick, one is the “correct” option that leads to better outcomes for the voters in the system, and the other is “incorrect” leading to not as good or even bad outcomes. Each agent in the system will have a set competence. This competence represents how likely they are to pick the correct option. The lowest competence is 50% which represents a voter who has no knowledge about an issue so would just be making a guess of which of the two options to pick. The highest will be a hard limit set around the 90-95% point, representing very well informed voters who are experts with the vote issue, but still capable of (rarely) making mistakes.

The competence an agent has will influence how likely they are to be a Guru or a Delegator. Agents with low (equal or near to 50%) competence should nearly always want to delegate to someone who has a higher competence and so a better understanding of the vote issue, however not always as there of course may be people in real life who don’t know about the issue but still trust themselves to vote more than anyone else. Higher competence voters will of course be less likely to delegate, having the knowledge to be able to confidently vote directly and be a Guru for other voters in the system, but they too may of course choose to delegate (with a lower chance) to a voter they believe is more competent than themselves. The issue is of course how do agents know the competence of other agents?

Agents in the model will in fact have two parameters based around competence, “Competence” and “Perceived Competence”. Firstly the “Competence”, this is the actual true competence of an agent, known only to the individual agent itself, as it can be sure of how much it knows about the issue whereas it needs to prove that to other agents. Then the “Perceived Competence” of an agent is how competent it is seen by the other agents that know it, this is an evidence based competence calculated by how many correct decisions the agent has made, it can also go no lower than 50% as the agents in the system should assume no agent is worse than randomly guessing (no intentional sabotage).

There are two types of runs the system will be able to do, “Continuous Learning” and “One Shot”, these are discussed more later, but in “Continuous Learning” the perceived competence will start at 50% for each agent will be recalculated after every vote, giving the percentage of correct decisions the agent has made or a min value of 50% as talked about above. However, in “One Shot” where each agent is only used once the “Perceived Competence” will always be equal to the true “Competence”, making the mathematically safe assumption that an agent far into using the LD system will have a perceived competence that has at least nearly evened out to its true competence.

To visually represent the perceived competence of a voter agent within the LD simulations (and the true competence in one shot simulations) the head colour of the agent will be changed to match its competence. Each agent will have a yellow head, but

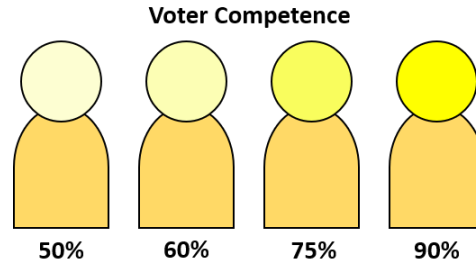


Figure 3.2: Competence Levels of Voters rising from left to right

at 50% competence it will be a very faded shade of yellow, the higher the perceived competence of the agent the more clear and bright the yellow colour will be (figure 3.2), so the voters that are seen as more competent within the system appear as “brighter” to the user of the simulation. Lower competence voters will be more common within the system, so there will be many with dim heads and few with bright heads, and in simulations where competence is important to voters it would be expected for the brighter head agents to commonly receive the most delegations.

Of course in the real world there are many different topics and issues to vote on, and different people will have different levels of knowledge on these, so the system could be more realistically designed having agents with many different competence levels for different vote topics. However, the assumption being made for this model is that having these different topics and competence levels in the system would just be the equivalent of running different individual tests with one competence level for all votes, so having one competence level generalises this for experimental purposes and should get similar results.

### 3.1.3 Power (and Visualised Weight)

When it comes to voting each agent that has chosen to be a Guru has a power. That power is equal to 1 plus the number of delegated votes they have received, with the 1 representing their own vote. So, when a Guru votes for one of the two options the number of votes for that option increases by 1 multiplied by the power of that Guru, with the lowest possible value being 1, which is the power of a Guru that receives no delegations.

In this model Delegators always have a 0 power as they do not vote, while they did have the option to vote and they may have received delegations that they transitively delegate to another agent, they do not use those votes themselves, so in the current vote have no direct power. You can look at Delegators in this model as having a “Potential Power” in the continuous learning simulations as they may decide to stop delegating on the next vote and then have power equal to at least one, in one shot simulations this is irrelevant as each agent is only used once so they will never have power in a vote.

What all agents do have in the model is a weight, which is considered different to power in this model of LD, the weight is a measure of their influence in the system, being how many votes they control, either to vote directly with or delegate. All agents have a weight of at least one, deciding what to do with their own vote, like power the weight of an agent is 1 plus the number of received delegations.

In the visualised simulation the power and weight of each individual agent is represented by altering the agent model, the weight is shown by increasing the size of the

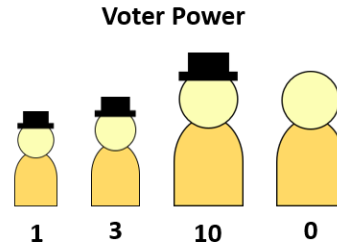


Figure 3.3: Power of Agents displayed underneath models of increasing size representing more received delegations

agent (figure 3.3) making them visually look like they have more weight. In figure 3.3 from left to right the agents have received 0, 2, 9, and 9 delegated votes respectively, however the powers listed below show the rightmost agent as having power 0. This is because agents will only have power if they are a Guru, and visualised wearing a Guru hat, so even though the two bigger agents have received 9 votes via delegation each, the Guru agent has a voting power of 10, and the Delegator has 0 vote power.

### 3.1.4 Preferential Attachment and Competence Importance

Agents that chose to be Delegators have to choose which other agent they want to delegate their vote to. While how delegations will work in the model are discussed in more detail later, the factors agents consider will be covered here. The idea of the simulation is to best represent agents as real people sharing the same values as those people. The two key factors chosen for this model are based on two values, the first factor, Preferential Attachment, is based on the popularity of agents, and the second factor, Competence Importance, is based on the knowledge of agents.

A Preferential Attachment process is where some quantity such as wealth or followers, is distributed among a number of individuals or objects according to how much they already have. It is commonly concerned with network growth, with a lot of work recently looking at how social networks form. The first application of preferential attachment was given by Price in 1976, who referred to the process as a "cumulative advantage" (Price, 1976), meaning agents in a network who already had a lot of the quantity would be more likely to gain more of it.

In the case of this Liquid Democracy model, the network is the transitive network of delegations, and the quantity that agents accumulate is delegations. Agents will have a preferential attachment parameter, this parameter indicates how important the number of delegations an agent has is when choosing who to delegate to, if set to zero it does not factor into the decision, but set to higher numbers and the agent with the most delegations already is likely to be chosen again. The idea here is that when preferential attachment is considered in the LD context, popular agents will gain more delegations for being popular while they may not be the best option, and the growth of this delegation network as more delegations are made could lead to super voters, who amass a huge voting power.

Competence Importance however favours knowledgeable agents, where more competent agents are more likely to gain delegations. As will be covered in the delegation section agents will only ever delegate to other agents who are perceived as more competent than themselves, but the agents will also have a competence importance parameter

that determines how important higher competence is when delegating. In the case competence importance is set to zero, then an agent who is seen as 90% competent isn't preferable to one of lower competence, e.g. 55%, however setting it to higher values means higher competence agents are much more likely to be delegated to than lower ones. Issue knowledge is important when picking a representative, a higher knowledge would typically mean a better representative to delegate to, however the issue of super voters may well come up again, e.g. the highest perceived competence agent getting the majority of delegations.

Both varying the importance of popularity and competence (individually and together) in delegation decisions should can to some interesting experimental results from the simulation when analysing the effects these might have in implementing the LD system in a real community of voters.

## 3.2 Liquid Democracy Environment

Now how agents will be modeled has been detailed, this subsection will cover how they interact with the LD system, looking at delegations, networks, and voting, as well as how all these parts of LD will be visualised to the user in the simulation program.

### 3.2.1 The Visualised LD Network

The Liquid Democracy voting system will be visualised using an environment consisting of three areas, shown in figure 3.4. The first of the three areas shown at the bottom

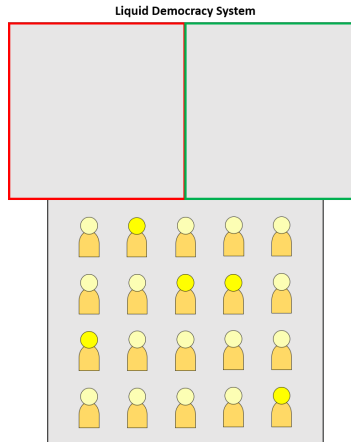


Figure 3.4: Design of Liquid Democracy Voting Environment

of figure 3.4 is where the voter agents described above will be spawned into the environment. See in the example figure that 20 voters are in the environment with varying competence levels shown by their heads “brightness”. This spawn area is where the agents will be positioned before every vote and will make their delegation decisions. The second and third areas shown at the top of figure 3.4 are the voting areas, the voters will move to one of these two positions to show their support for that option, there are always two areas due to the LD model using a binary voting system.

### 3.2.2 Local Networks

In the community using the LD system it would be unreasonable to assume that all voters would know all other voters, so a parameter that will be settable in the LD system is the “Network Size”, setting how big the local network for each voter is (how many other voters it knows). Setting the network size to  $N$  means each agent will know  $N-1$  other agents (as the agent itself is included in the local network). The lowest the



network size can be is 1, which will make the LD system operate like direct voting, as the agents are only aware of themselves so have to vote direct, the highest it can be set is to the number of voters in the community, e.g. 20 in Figure 3.4, meaning every voter is aware of (and can delegate to if they are more competent) every other voter.

The local network of an agent will be the  $N-1$  closest voters to it in the starting area in a one dimensional line, figure 3.5 shows example local networks for three agents (A, B, and C), when the network size is set to 5. Agent A has two voters either side of it in its network and itself to make 5. In the case the network size is set to an even number then the number of voters added to the local network will not be equal with one side having one extra (e.g. in a network size of 6 there will be 2 voters from one side and 3 from the other), which side has the extra voter will be decided randomly.

Agent B in figure 3.5 shows what is meant by the local networks being one dimensional, the  $N-1$  closet voters are taken from the left and right looping up and down rows, not the voters surrounding it in 2D space, so the local network for agent B loops to the row above. However, the end of the community of voters does not loop back to the start and vice versa, so if there are not enough local voters on one side of an agent they will take more from the other side, as shown for agent C in figure 3.5 taking all 4 of its neighbour voters from its left. This also means that the left most neighbour of agent A has the exact same agents in its local network as agent A does.

For the agents in the example of delegations shown in figure 3.6 agent 4 delegates to

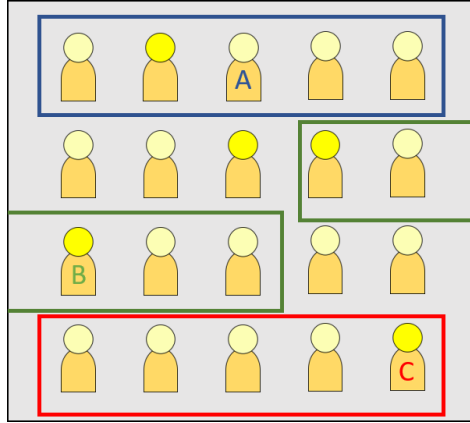


Figure 3.5: Local Network Sizes of Agents

agent 8. There are two possibilities for how this is possible, either the agents must have a local network size of at least 8, or agent 4 learnt about agent 8 through transitive delegations. In the continuous learning simulations if agents are assigned a Guru that isn't in their local network through transitive delegations they will add that Guru to their network as they now know about them, so then they can delegate directly to them in future votes.

### 3.2.3 Delegations

Delegations are the key component in any LD model. Before each vote each agent in the system must choose whether it wants to delegate its vote to another voter. In this model of LD each agent may decide to delegate their vote, although the higher their competence is the less likely it is they will. In order for an agent to see another voter as an acceptable option to delegate to that other voter must have a perceived competence higher than true competence of the agent, and it must be known in the local network

of the agent. It will be possible for an agent to choose that they want to delegate but then find out there are no acceptable higher perceived competence agents and so will be forced to vote directly (be a Guru).

The agents that choose to be Delegators will pick randomly between all the acceptable options, however the randomness may be weighted in favour of certain agents in the system. As discussed above agents will have a preferential attachment parameter, which when set to a number above zero will make them more likely to pick voters that have the most delegates already, this will much more effect agents that make their delegation decision after many have already been made (e.g. in figure 3.6 if delegations were made in number order agent 10 would have been heavily biased to delegate to agent 8 over the other acceptable options). The other parameter is competence importance, where agents will be more likely to delegate to higher competence voters.

An example of what the visualised delegations should look like is shown in figure 3.6, with delegations shown with a clear line between agents, with arrows in the direction from Delegator to delegate. Figure 3.6 shows the same system of 20 agents spawned in figure 3.4, there are 20 voters, 10 have chosen to be Delegators, and 10 have chosen to be Gurus as shown by the hats, of the 10 Gurus only 4 of them received delegations, so the others just vote directly with their own vote. Agent 8 has the most delegations, visualised by it being the biggest, with 5 in total, 4 direct and 1 transitive, the transitive vote is from agent 10 that is given to agent 9 who then gives 2 votes to 8, 9 is slightly bigger due to the 1 delegation but has no power as a Delegator, agent 8 has power 6.

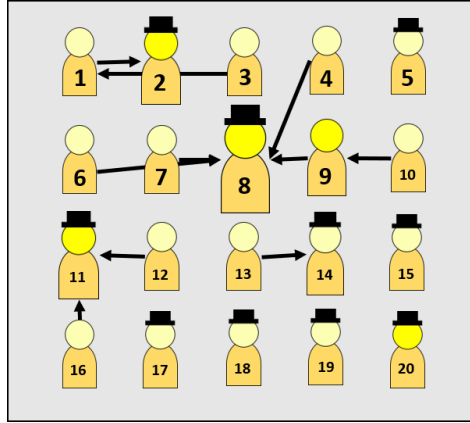


Figure 3.6: Design of Delegations Decisions Example

Other Gurus that have received delegations in figure 3.6 are agent 14 with one delegation, and agents 2 and 11 with 2 delegations each, but where 11 has 2 direct delegations, 2 has 1 direct and 1 transitive. Note that although higher competence voters are more likely to receive delegations (if competence importance is set above zero) it is not guaranteed, shown by the higher competence voter, agent 20, having received no delegations, where a lower competence voter, agent 14, has received one.

### 3.2.4 Voting and Results

In the LD model once all the agents have been created (or returned from their previous vote) and have made their delegation decisions the voting on the current issue will then begin. As stated the voting system that will be used in this LD model is a binary voting system, where there are two options with one being correct and the other being incorrect, with the correct option being the one that results in a better outcome for the

community using the LD system.

The voting system will be simple, of the two given options one of them is randomly assigned to be the correct one, each Guru chooses an option with their chance of choosing the correct one being their competence level (at minimum 50%), and the chance of choosing the incorrect one being 100% minus their competence level (at maximum 50%). When a Guru picks an option, that option gains a vote for the Guru and for each of the Delegators of that Guru (the number of votes for the option rises by 1 times the power of the Guru). After all the votes are collected the totals of the two options are compared, and the winning option is announced to be either the correct or incorrect option. The model will keep track of how many correct (and incorrect) decisions have been made by the community.

In the uncommon case that the total votes for both options add up to the same amount a coin flip mechanism will be employed to choose the winning option, so for draws half of them should result in correct decisions and half incorrect decisions.

The visualised LD simulation will display this voting procedure by having each Guru vote one at a time by starting to move to their desired position, the left of figure 3.7 shows the first Guru has made its decision as it has started to move toward the vote option area on the right. When a Guru votes if it has any Delegators they also are choosing a position by proxy, so when the Guru moves towards its chosen vote option area the Delegators will move with it, as shown on the right of figure 3.7 as some Gurus

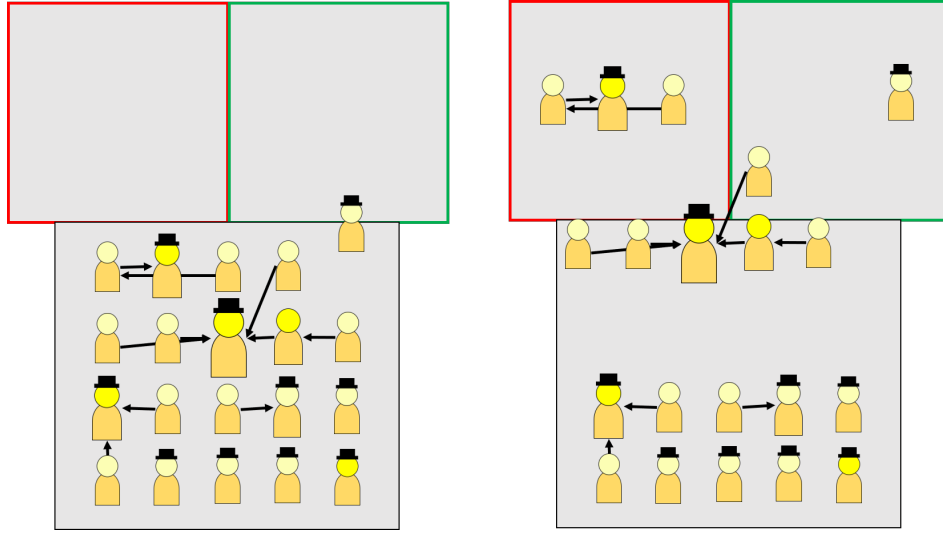


Figure 3.7: Voting Process Design with Moving Agents

with Delegators make their vote.

Once all the Gurus have made their votes and all the agents are positioned in the voting area of the option they have chosen, the result of the voting will be displayed to the user of the simulation, giving the number of votes for each option so it can be seen which one has won (figure 3.8 (left)). Then once the winning option is known it will be announced whether the option that won was correct or not by displaying the word “correct” or “incorrect” (the correct option happened to be the left one and won in figure 3.8 (right)). The total number of correct and incorrect options will be collected as the simulation runs for the totals to be displayed to the user at the end.

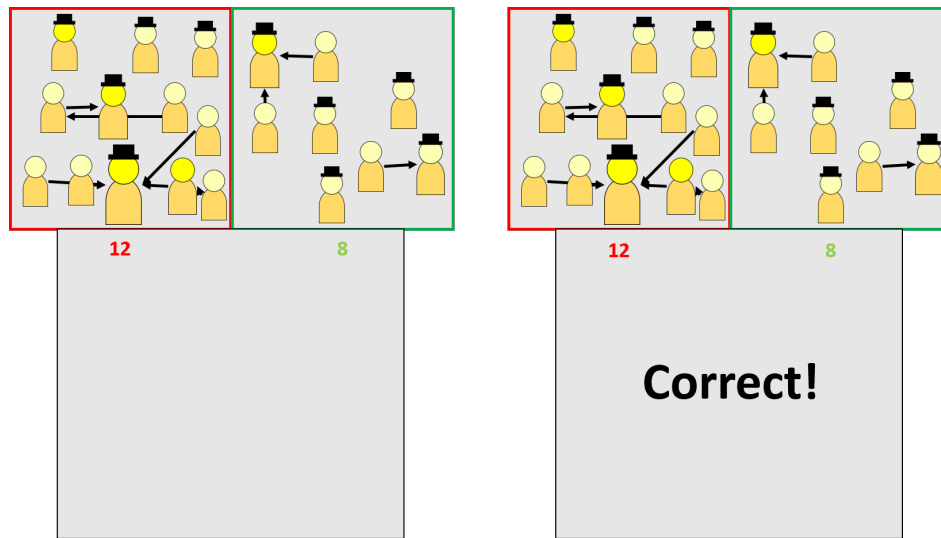


Figure 3.8: Result of Vote and Correct Decision

### 3.3 System Processes

Through the design section so far two different types of simulation have been referred to, “One Shot” and “Continuous Learning”. The main difference between the two is that a different set of agents are used for every vote in one shot, where in continuous learning the same agents are used for every vote and they “learn” more about each other as the rounds go on. This section will cover all the differences in detail and describe the full design of the processes for running both of these types of simulation.

#### 3.3.1 One-Shot Votes with Random Agents

The flow chart in figure 3.9 shows the chain of events that will happen when a One Shot Liquid Democracy Simulation is run by the user. A random set of  $X$  agents with varying competence levels, where  $X$  will be set by the user before running, will be spawned

into the system, their perceived competence to other agents that know them will be set to their true competence, simulating a LD network that has been running for some time so the two values will have evened out. Then each agent will make the decision to be a Guru or a Delegator, based on their competence, and in the case they choose to delegate they will choose an agent with higher competence to delegate to. If the user has chosen to visualise the simulation then Guru agents will gain hats and Delegator agents will have arrows going to their delegate as demonstrated in figure 3.6.

After all the agents have made the Guru or Delegator decision the voting will begin. Each Guru agent will vote for one of the two options with the chance of them choosing the “correct” one being equal to their competence, with that option gaining a number of votes equal to the power of the Guru ( $1 + \text{the number of Delegators}$ ), in the visualised simulation this will be shown by having the Gurus move to the vote area for their chosen option with their Delegators (figure 3.7). After all the votes are cast the option with the highest number wins, in the case of a draw the winning option is chosen at random with equal chance, and then it is revealed if the option was the correct one or not (displayed to the user in the visualised simulation like in figure 3.8).

Once the vote is over one of two things can happen depending on the number of rounds of voting have occurred. Before the simulation is run the user will have set how many rounds the simulation should run for which sets how many votes there will be. In one shot simulations in the case the desired number of runs hasn’t been completed at the end of the vote then as shown in the flowchart in figure 3.9 all the agents are



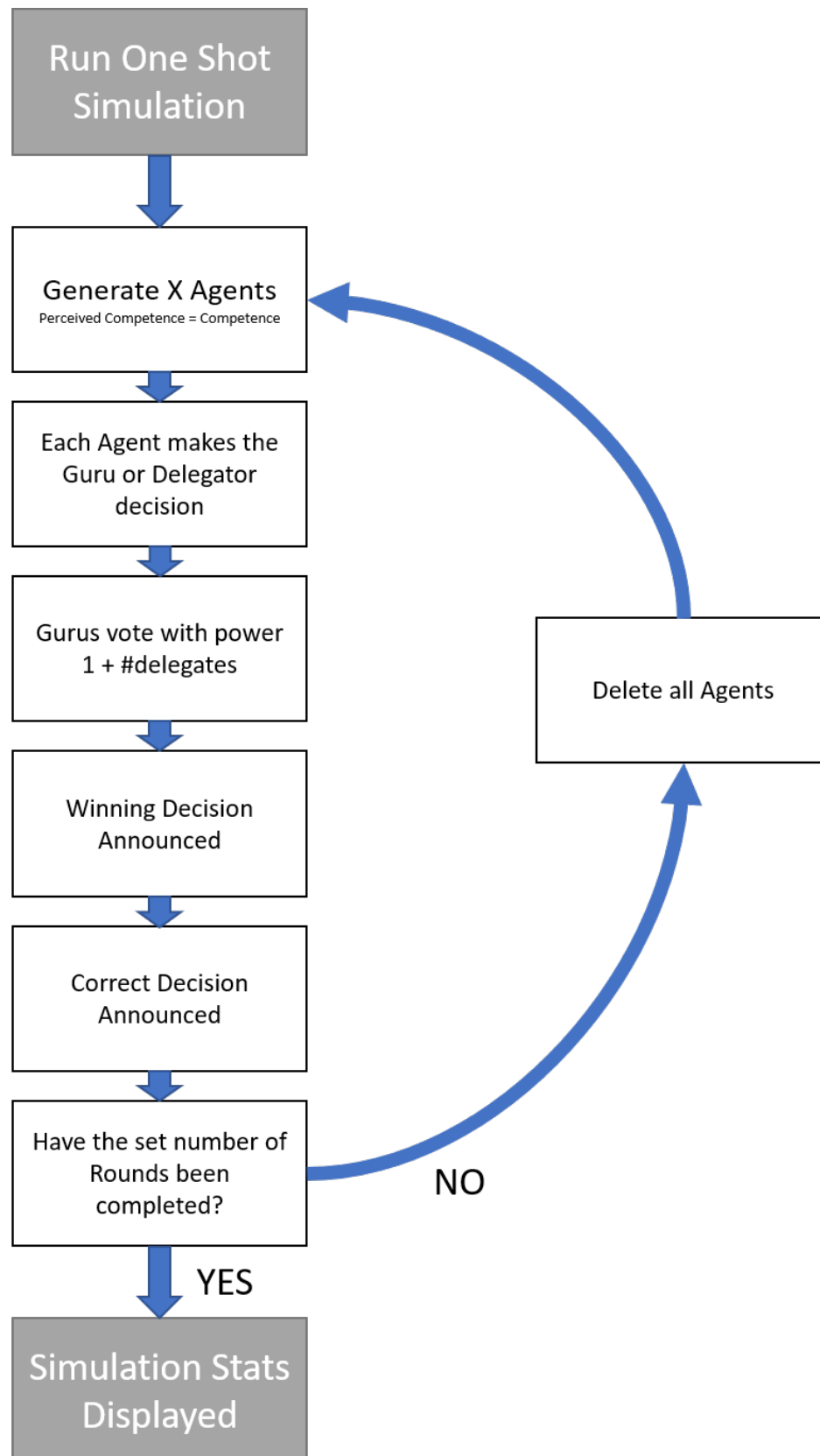


Figure 3.9: One Shot Simulations Flow Chart

deleted from the system and replaced with a new set of random agents and the process starts again. The voting with a new random set of agents is repeated until the number of desired rounds have been completed, if the desired number have been completed the simulation ends and the user is taken to a stats screen, showing the number of correct and incorrect decisions made in the simulation along with other stats.

Delegation cycles are an issue that need to be dealt with in LD models. This is when you have a delegation chain that loops from an agent back around to that same agent, so there is no Guru to actually cast the vote. However, in the model of these one shot simulations delegation cycles do not need to be dealt with due to the assumption that agents will only delegate to agents with a perceived competence higher than their competence and perceived competence always being equal to true competence, as this means there can never be a cycle, as there is a one way allowed order of delegations based on competence.

### 3.3.2 Continuous Votes with Learning Agents

The continuous learning simulations begin nearly identical to the one shot simulations, with the only difference in the setup being all the agents start with a perceived competence of 50%, which for many will not be equal to their true competence. Figure 3.10 shows the flow chart for the continuous learning simulations, like at the start of one shot the agents are spawned in, make the Guru or Delegator decision, the Gurus make their vote, the winning option is announced and then it is announced whether that was the correct or incorrect choice.

However after the first vote the difference of continuous learning to one shot is clear, only one set of agents is used for the whole simulation, agents are not deleted and replaced after every vote. This is to simulate a single community of agents using the liquid democracy system for the very first time, and the progress they make through using it, which is why all agents start with the same perceived competence of 50% as they haven't yet learnt which voters are good at making vote decisions.

So after each vote if the desired number of rounds haven't been completed agents will review the previous vote (see this on the right branch of the flow chart in figure 3.10), the perceived competence of each Guru will be updated, increasing if they got the last vote right and decreasing if not, it will be equal to the percentage of decisions they have got right (or at minimum 50% if it would be below that as no agent is seen worse than guessing). Then each Guru that made the wrong choice will remake the decision of being a Guru or Delegator, always with the same chance based on their true competence, and the LD model assumes Gurus who were right in their last vote will always remain as Gurus.

The agents who were direct Delegators to Gurus who made the wrong decision in the last vote will decide whether they want to keep them as a Guru. The chance they remove them as a Guru will be based on the perceived competence of the Guru, the higher the perceived competence the less likely the Delegator will remove the delegation. Going down the transitive delegation chains from the direct Delegators of wrong Gurus,

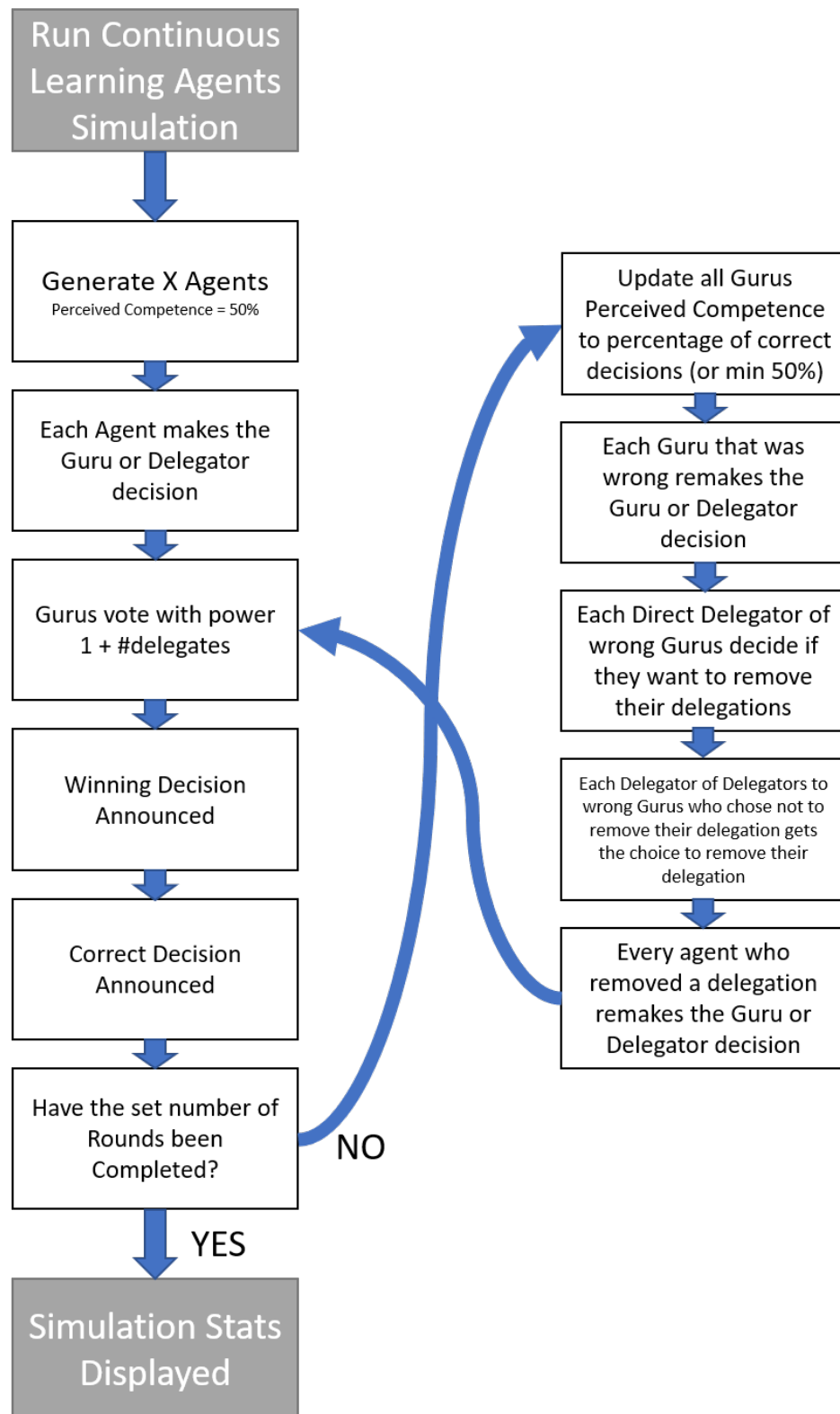


Figure 3.10: Continuous Learning Agents Simulations Flow Chart

if a Delegator decides to remove its delegation to the wrong Guru it is assumed that all the Delegators to that Delegator will keep their delegation, however if the Delegator decides to keep the delegation all of its Delegators will have a chance, also based on the Gurus perceived competence, to remove their delegation, and this continues down the chain.

Because of perceived competence not being equal to true competence there may be delegation made to agents who actually have lower competence than the Delegator, if a change in a Gurus perceived competence makes it lower than any of its Delegators true competence then those Delegators will always remove their delegation straight away. The LD model assumes that if a Guru is right then all of their Delegators will remain as Delegators. All the agents who removed their delegations due to having wrong Gurus will remake the Guru or Delegator decision, in the case they decided to delegate again it will not be made to any agent that results in them having the same Guru they had previously, if they have no acceptable delegations that don't result in them getting that agent as a Guru then they will become Gurus themselves.

In the visualised simulations after each vote (except for the last) the voters will all move out of the voting area back to their starting positions before reviewing the vote and making delegation decisions for the next vote as described above.

After a number of runs the perceived competence of agents should get very close to their true competence. This simulates voters in the system learning how knowledgeable

the potential Gurus in the community are when it comes to voting through previous experience as having them as a Guru or through word of mouth in their local network. Delegations and agents who are Gurus should become more stable as the community becomes more experienced with the LD system, with all Delegators delegating to Gurus who are truly more competent than them. Again after the desired number of voting rounds are completed a statistics screen for the simulation will be shown to the user.

Unlike in the one shot simulations the problem of delegation cycles will have to be handled as the perceived competence of an agent will not always be equal to its true competence. An example being if an agent A has a higher competence as an agent B, but A perceives B as more competent than itself as B has by chance made a high number of correct decisions, and B perceives A as more competent than itself for the same reason, then they may both want to delegate to each other. This will be handled with a very simple system of not allowing delegations to be made if they result in a cycle, which in a real world LD delegation system would be achieved by not listing any of the voters who would cause a cycle to the user to delegate to. While this is very simplified it deals with the problem, the possibility for cycles should also quickly reduce as perceived competences become more accurate.

### **3.3.3 Visualised Simulations: Auto Run vs Analysed Run**

A key part of the liquid democracy system being designed is that users will have an option for the simulations to be visualised so they can see the agents using the voting system. The visualised simulations will have two running options, an auto run option

and an analysis option. The auto run options will just have the simulation run from start to finish with no input from the user after pressing the initial start button until it reaches the statistics screen. Users will be able to see the agents spawn in, make delegations with delegation lines, and move around to vote, but will have no control over the simulation.

The analysis mode will give the user a bit more control over the simulation. The key point of visualising LD is to increase understanding, and the analysis mode aims to make it easier to understand. In analysis mode the simulation will stop after each action, so it will stop after all the agents are created, it will stop after each set of delegation decisions, and it will stop after each vote. The user will have control of when the simulation resumes, for example by pressing a key or button, this gives the user more time to analyse the current state of the system before moving on to the next step, being able to see the competence of each voter, how many Gurus there are, look at all the delegations, and may help them see why some votes go wrong (e.g. a super vote making a mistake).

### 3.4 Systems UIs

The simulation software has a number of UIs, namely the start menu for setting up the simulations and statistics screen for analysing the results from them.

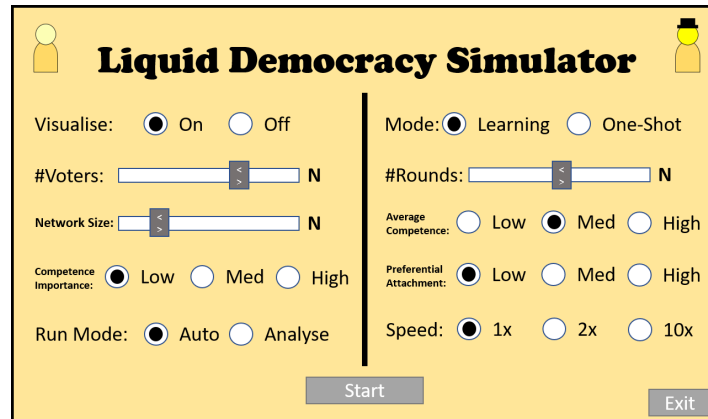


Figure 3.11: Design of Simulation Software Start Menu

### 3.4.1 Setup Menu

When the program is launched the first thing the user will see is the setup menu (design in figure 3.11) which will present them with a number of options to configure a liquid democracy simulation. The two options (Visualise and Mode) at the top will control which type of simulation to be run, the visualise option on will give a visualised simulation for the selected settings with each agent, delegation, vote, and outcome rendered and shown to the user as illustrated by all the designs through this section, whereas the visualise being off will run all the simulations in the background with no visuals of the processes for the user, which should mean it will complete a lot quicker. The mode can be changed between continuous learning and one shot which has just been explained in the last subsection.

Many of the other options of the setup menu allow the user to adjust the parameters that have been discussed through this section, this includes options for the LD commu-



nity such as how many voters there are, how many rounds of voting they will do, and how big their local networks are. It also includes options for agent parameters, such as how high the average voter competence is, how important they find competence, and how much they are swayed to delegate to already popular agents (agents with more delegations).

Finally users can set the run mode to auto or analyse, and the speed the simulation will run at (this is only for the visualised simulations as the non-visualised (or console) based simulation cannot be analysed and will always run at the same speed faster than the visualised counterpart). There will also be clearly labeled buttons to start the simulations and exit the program.

### 3.4.2 Agent Information UI

During the visualised simulations users are able to click on an individual agent to get information about just that agent, such as if they are a Guru or Delegator, their power, their competence, who their Guru is, their Delegators, and how well they have voted so far. A design for this is shown in figure 3.12, with the UI coming up on the side of the screen with the simulation and selected agent on the left, users will be able to close and open it at any time. This UI should make it clear to users how an individual agent is acting within the system, and make it clearer who is delegating to who in big community simulations with lots of agents.

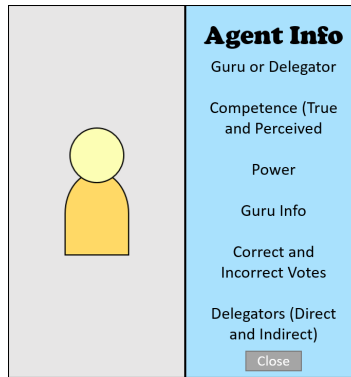


Figure 3.12: Design of Agent Info UI that can opened in simulations

### 3.4.3 End Results Menu

After every simulation is run the user will be presented with a statistics screen (design shown in figure 3.13), which will allow the user to analyse how well the community did within the simulated LD system. The screen will show some of the setup menu settings to remind the user how they set it up, and the key idea behind how well a simulation went will be shown by the number of correct and incorrect decisions made. The more correct decisions the more successful the community was.

Some additional stats will be shown (as displayed in figure 3.13 on the right), such as the average number of Gurus there were each vote, the average power of the highest powered Guru (which could give an idea of if there were commonly super voters), and the average vote competence which gives the average competence behind each vote made (the vote competence for each agent is given by their Gurus competence so this value should indicate if good delegation decisions were typically made). Then there will be a back button which will return the user to the setup menu where they can run

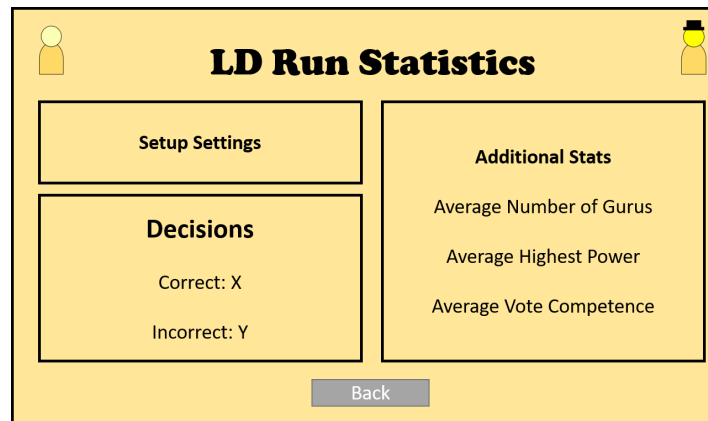


Figure 3.13: Design of Simulation Software Statistics UI

another simulation or exit the program.

# Chapter 4

## Liquid Democracy System Implementation

This section will discuss the implementation of the Liquid Democracy model designed in the previous section into a working software system.

The system was created using the Unity game engine, with the programming language being C#. Early in the development process Python was experimented with to make a basic LD program, chosen for its mathematics libraries (e.g. numPy) as it would be useful for making gamma and logarithmic functions needed to model different aspects of the simulations. The created Python program produced a static visualisation of the LD system after a vote, see figure 4.1, in the form of an actual graph, with the voters as nodes, delegations as directed arrows, weight represented by node size, with green representing “yay” voters and red representing “nay”.

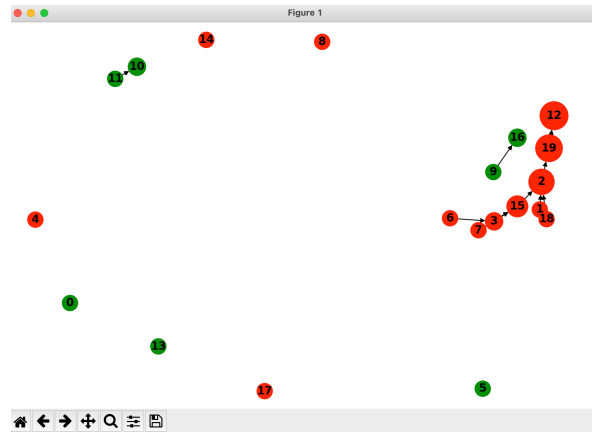


Figure 4.1: Original Python Visualisation

Eventually the decision was made to instead use Unity with C# because Unity is a game engine that allows for the easy creation of 3D environments. This meant it allowed for a much more detailed visualisation that users should be able to relate the simulation more closely to the real world problem, when compared to what was possible with a 2D graph which could be hard for users to understand. It also meant it was possible to model components of the system after their designs (e.g. the Agents look like their design in figure 3.1). As stated Python was originally chosen for its mathematics libraries, but libraries were found in C# that provided all the needed functionality (creation of gamma and logarithmic functions).

A video demo of the working LD Unity implementation is given in appendix B.

## 4.1 Visualised Liquid Democracy

First, this subsection shows how the created software visualises the liquid democracy simulation to the users, looking at the created Unity environment and the components acting within it. Then the more technical details of the underlying calculations in the C# code that controls the environment and the agents will be explained in the later Simulation Processes subsection. The figures showing the visualisation of the components should make that section easier to follow.

### 4.1.1 Environment

Following the planned design of the LD environment in figure 3.4, the simulated environment is very simple (figure 4.2), being made up of 5 cubed areas. The biggest area at the bottom of figure 4.2 is where the agents are spawned into the system, and it is where they stand between every vote and where they make their delegation decisions. Then there are two bridges crossing over to two voting areas. The voters will stand in one of those two areas to signify their vote. There are the two letters A and B each above one of the voting areas in different colours (blue and pink like the bridges). This represents the two different options, and can be looked at as policy A or B, candidate A or B, or simply “yay” or “nay”, as the LD voting system is binary.

The environment is controlled by an empty object named “Liquid Democracy” which itself contains a C# script called “LiquidDemocracy.cs”. Using the script this object creates the agents and spawns them into the environment, tells the agents when to

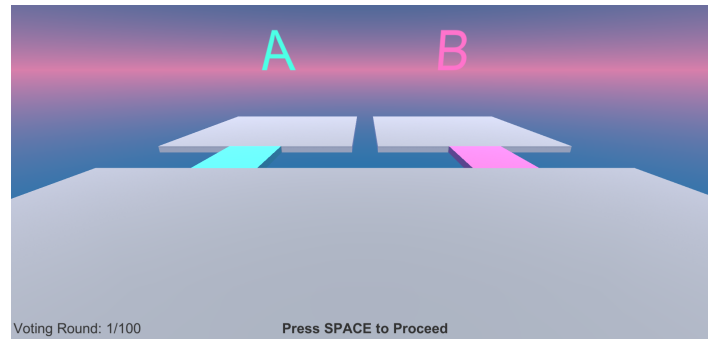


Figure 4.2: Liquid Democracy Simulation Environment in Unity

make their delegations decisions, and tells the Gurus when to vote. The settings chosen for the simulation on the start menu UI by the user are carried through to the Liquid Democracy script from another empty object called “Stats Manager”. The “Stats Manager” also counts the number of correct and incorrect decisions made in the environment as well as other statistics, but this will be covered more in the Simulation UI and Statistics subsection.

A skybox with similar colours to the two option colours surrounds the environment, and there is a UI overlapping the camera that shows the user the environment. The first part of the UI shown in the bottom right of figure 4.2 simply shows how many votes have taken place out of how many in total were set to be run.

By default the user has no control over the “Liquid Democracy” object controlling the simulation environment as it will be set to auto run. However, if they select the analyse mode then after certain events the prompt shown in the bottom middle of figure 4.2 will be displayed with the environment pausing until the user presses the spacebar,

at which point the prompt will disappear and the simulation will continue. The events that will cause the simulation to pause are all the voters being spawned in (or getting back to their start position after a vote), all the voters making their delegation decisions, all the voters voting, and the results being announced. This gives the user the chance to analyse the current state of the system (competence levels, and delegations) before the simulation moves on.

Voter agents are spawned into the environment at the start of every vote in One Shot mode, and just once at the start of continuous learning mode. There can be a minimum of 5 and a maximum of 100 in the visualised simulation. How they are positioned in the environment depends on how many they are. Between 5 and 29 they will be in rows of 5 (figure 4.6), between 30 and 59 they will be in rows of 10, and between 60 and 100 they will be in rows of 20. When they are in rows of 5 and 10 the camera is positioned as shown in figure 4.2, but for bigger sets of agents when they are in rows of 20 the camera will be further out to fit them all in (see figure 4.3 with 100 agents in the environment), showing a further out view of the environment.

### 4.1.2 Agents

Each voter is represented in the simulation using an individually rendered agent modeled closely to the planned design. The voter object was built in Unity itself, see the right of figure 4.4 to see the individual components listed in Unity. The voter object consists of a head made of a sphere, and a body also made of a sphere but just stretched vertically. The head also consists of two smaller spheres to give it eyes to make the



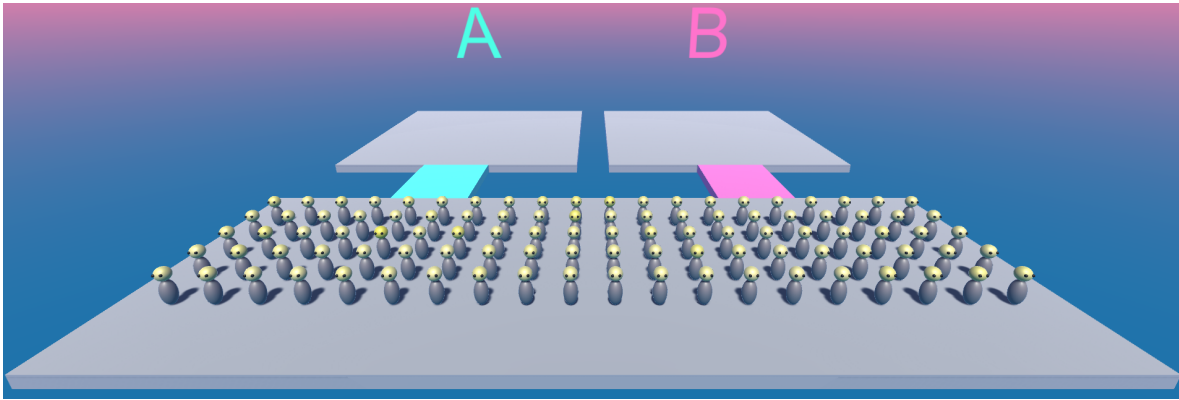


Figure 4.3: 100 Agents Spawned into Environment

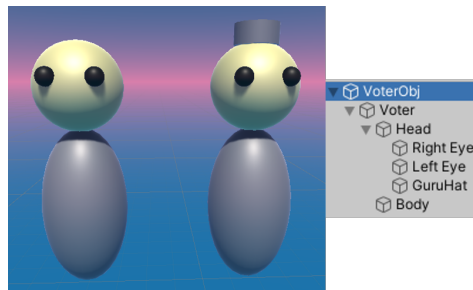


Figure 4.4: Voter Agents in Unity, Left: Delegator, Mid: Guru, Right: Unity Object Build

model better resemble a real person, making what they are meant to represent more recognisable to users. They also can have a hat to signify they are a Guru; no hat means they have just spawned or are Delegator. Both the Delegator and Guru models are shown on the left of figure 4.4.

Each voter is controlled by a different copy of the same script, called “VoterHandler.cs”. This contains all the voter functions, such as choosing to be a Guru or Delegator and casting votes, and also contains their parameters such as their individual



Figure 4.5: Gurus with Different Colour Guru Hats

voter ID, who their Guru is, and their competence, as well as parameters indicating how important potential delegate perceived competence (competence importance) and popularity (preferential attachment) are to them. Also each voter has a randomly assigned colour. This is the colour given to their Guru hat (and to the delegation lines that go to them as the delegate) to make each Guru distinguishable (figure 4.5).

Each voter's perceived competence is visualised by the shade of yellow their head sphere is. In one shot simulations this will also represent their true competence, whereas in continuous learning it represents their personal correct vote percentage. The colour of the heads are changed by taking the Unity material component from the head sphere object and changing the colour of that based on their perceived competence parameter. Figure 4.6 shows a set of 20 newly spawned voters. It is a one shot simulation so the varying head colours represent their actual competence: the pale yellow heads are low competence, the darker the shade of yellow the higher the competence.

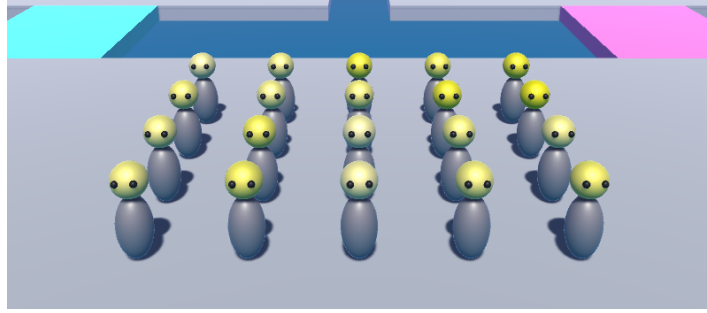


Figure 4.6: Visualised Perceived Competence with 20 Spawned Voters

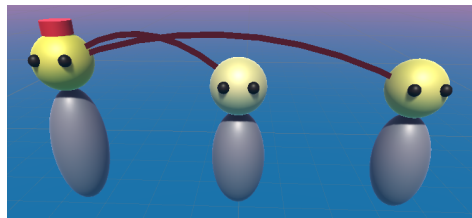


Figure 4.7: Delegation Lines Example, one Guru receiving two delegations

### 4.1.3 Delegations

Delegations are represented with a line going from Delegator to delegate, figure 4.7 shows two Delegators delegating to the same voter who has chosen to be a Guru. Delegators will only delegate to voters with a perceived competence higher than their own true competence as can be seen to be the case in figure 4.7 with the Guru receiving the delegation having a darker shade of yellow colour for their head indicating their higher perceived competence.

The colour of the delegation line will be the same as the hat of the Guru it goes to, either directly or transitively. Figure 4.8 shows the same set of voters spawned in figure 4.6 having now made the decision to be a Guru or Delegator, and the Delegators have

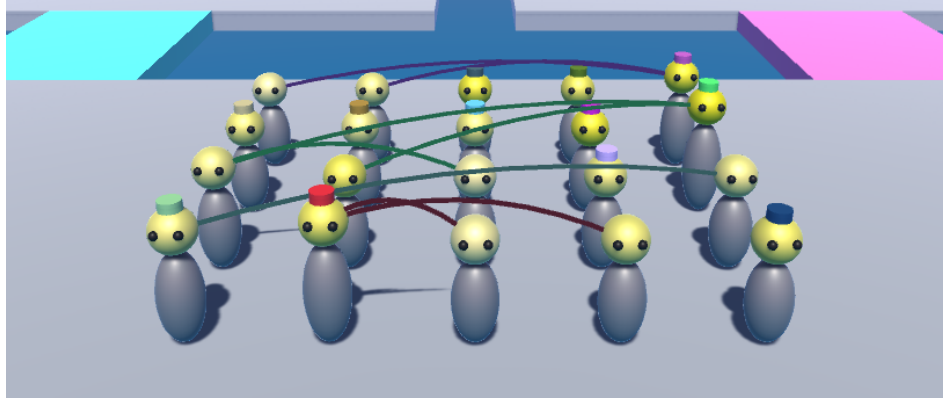


Figure 4.8: Delegations shown in Unity Simulation Software with 20 Voters

chosen who to delegate to, with different delegation line colours going to the Gurus with the matching Guru hat colour. The delegations from figure 4.7 can be seen in the front row of the voters in figure 4.8, with two red delegation lines going directly to the red hatted Guru. There is also an example of transitive delegation with the lime green hatted Guru receiving 3 delegations, 2 directly, and 1 transitively. Figure 4.9 separates just the voters with this Guru for easier viewing.

In figure 4.9 the rightmost of the 3 Delegators delegates to the leftmost agent, which is possible as the leftmost agent visibly has a higher perceived competence, but the leftmost agent also chooses to delegate, delegating directly to the lime green Guru, so the original Delegator transitively delegates to that Guru to and the line changed to the lime green colour. If the leftmost agent decided to stop delegating then the delegation line colour going into it from the rightmost Delegator would change to whatever its Guru hat colour is. The use of colours of the delegation lines removes the need for arrows as a user can scan from a given agent back to any of its Delegators using the

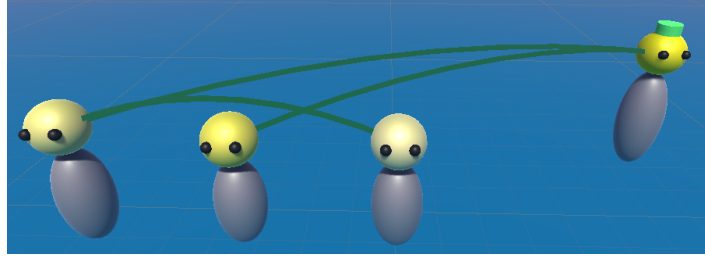


Figure 4.9: Example of Transitive Delegation

colour of the lines, and means you don't have to locate an arrow direction to understand the order and direction of a chain.

Agents make the decisions to be a Guru or Delegator based on their competence, with the higher competence agents being more likely to be a Guru. As the agents in figure 4.8 have perceived competence equal to their true competence it shouldn't be surprising to see it is mostly, but not only, the higher competence agents have chosen to be Gurus. As stated in the design and as is the case in the simulation featured in figure 4.8 the decision of who to delegate to is based on factors such as perceived competence and popularity of the other agents, so again it should not be surprising to see the higher competence more dark yellow headed agents getting the most delegations.

Although not visible in the simulation itself the agents have a set local network of agents they can delegate to as specified in the design. The agents in figure 4.8 have a local network size of 5 like in figure 3.5 from the design section, limiting who they can delegate directly to. For example the agent at the top left could not delegate directly to the agent at the bottom right.

The number of votes an agent has delegated to them is represented by the size of the agent: the more it receives the taller the body of the agent model becomes. In figure 4.9 the leftmost Delegator is taller than the other two as it receives 1 delegation, meaning it has the weight of 2 votes that it controls where the other two only have a weight of 1. Agents are said to have power only if they are a Guru, so without a Guru hat they have zero power, and with a hat their height represents both their weight and power with their power being 1 plus the number of received delegations. The agent in figure 4.7 has power 3 and is taller than its two delegates.

For an extreme example of height representing power see figure 4.10 that shows a Guru in a community of 100 voters that has received 34 delegated votes, having a power of 35 and towering over the other agents. This voter in figure 4.10 is an example of a super voter: it has amassed a large amount of votes that it could use to swing the vote result either way.

The delegation lines are their own object within the Unity environment. Each voter has a “delegationLine” parameter which itself has a “DelegationHandler.cs” script. This delegationLine object parameter is how the agents give the colour of their Guru to the line, and how they set the start position line as their head and then end position of the line as the head of their Guru. The lines are constantly rendered by the Unity Update function to remain between the heads of the Delegator and delegate even when they are moving in the environment.

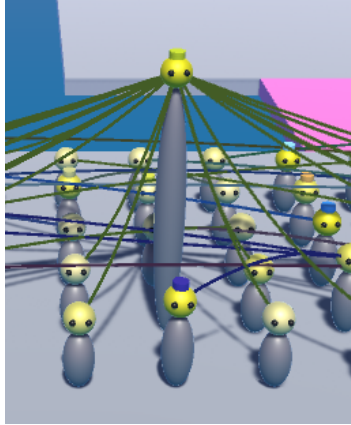


Figure 4.10: Super Voter Guru with 35 Power from 34 Delegations

#### 4.1.4 The Voting

The representation of the voting process is very simple. One of the two options is randomly chosen with equal chance to be the correct one. Then each Guru chooses an option with their true competence being the chance they pick the correct one. To show this voting process each Guru starts moving towards the voting area of their chosen option one at a time, with all their Delegators moving with them as a group (figure 4.11), the delegation lines remain between the agents to make the groups clear. As shown in figure 4.11 as the voters cross the bridges to their options their body changes to the colour of their selected option.

The voters move to the area by having a set path of positions consisting of the entrance side of the bridge of their chosen option, the exit side of the bridge, and then a random position in the voting area. The movement is handled in the “VoterHandler.cs” script changing the transform position of the voter object in the Unity Update function

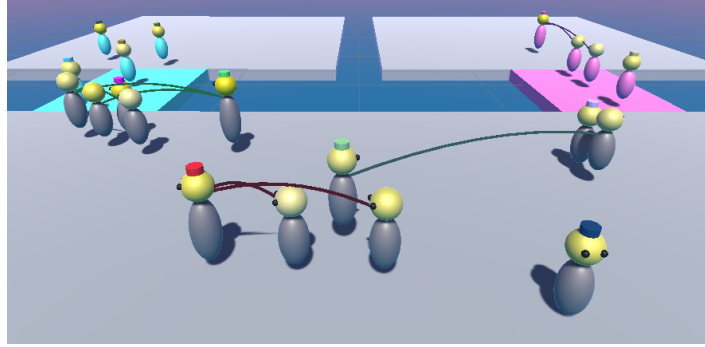


Figure 4.11: Voters Moving to Chosen Option Area to Vote

using the `MoveTowards` method. While moving the voters are animated to bounce up and down to make it look like they are actually making themselves move.

Once all the voters are in position in one of the voting areas the number of voters in each area is displayed where the A and B used to be to show which option wins (figure 4.12). In the case of a draw one of the options is randomly chosen to win with equal chance, simulating a coin flip. The agents who are in the winning option area will celebrate by jumping up and down and the agents in the losing option area will put their head down and shake it, figure 4.12 shows the agents mid animation.

After the total votes for each option have been revealed which options is right and which is wrong is revealed above the voting areas, at which point the total number of correct and or incorrect decisions is incremented by the Stats Manager that will be displayed to the user at the end of the simulation. If the correct option won the vote all the agents will celebrate with the “win” animation, else all the agents will be disappointed with the “lose” animation, in the case of a draw a user can tell which



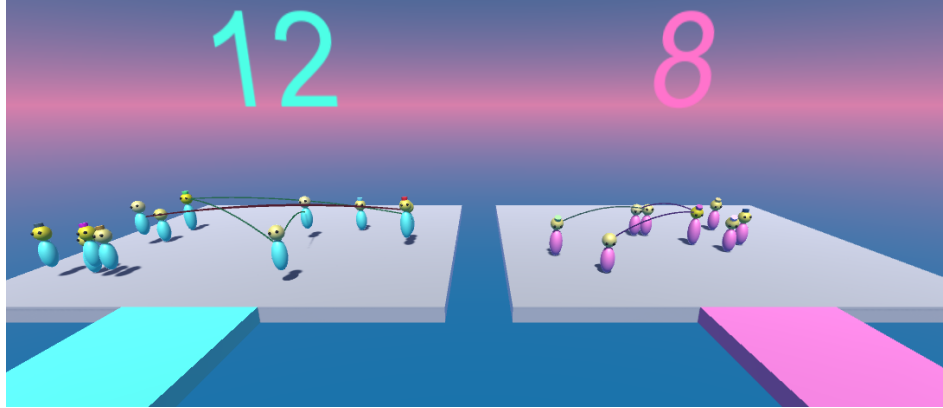


Figure 4.12: Results of Vote, displayed in created Unity Simulation

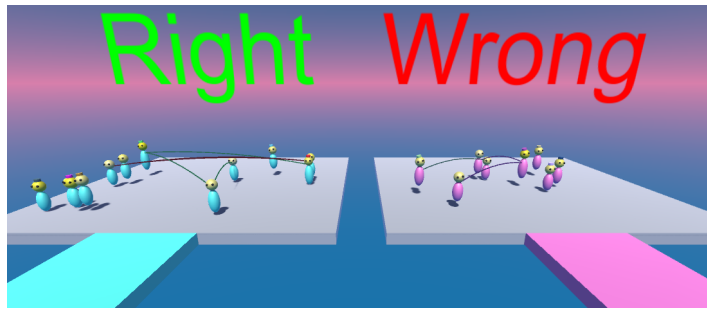


Figure 4.13: Correct and Incorrect Options of Binary Vote Displayed

option won from the animation of the agents.

#### 4.1.5 Animations

Just as the agent models were created for this project the animations for the agents were also newly created. Each agent object has its own animation controller attached with a graph of animations (figure 4.14). Upon creation of an agent the animation controller starts in the idle phase. They are then sent different triggers to start different animations. When an agent starts moving they are sent a “Move” trigger to do their

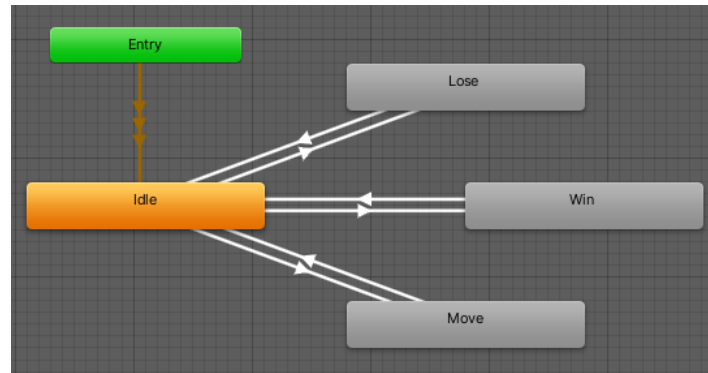


Figure 4.14: Animator Controller of Voter Agents Built in Unity

moving animation; this loops until they are sent a trigger to stop and go back to idle. In the case of their vote winning or the correct option winning they will be triggered to do their “Win” animation, in the opposite cases they will be triggered to do their “Lose” animation, these animations end and go back to idle after a set time.

All 3 of the animations were created in Unity, see for example the “Lose” animation being made in figure 4.15. The animations are created by positioning the parts of the voter at different positions at different time stamps, with them moving between those positions in the time set. In the case of the “Lose” animation their head is set to go down and then shake from side to side. The reason the voter object itself is positioned inside an empty object called “VoterObj” (see right of figure 4.4) is that it allows it to be animated in its own local space instead of teleporting to the middle of the global space when animating.

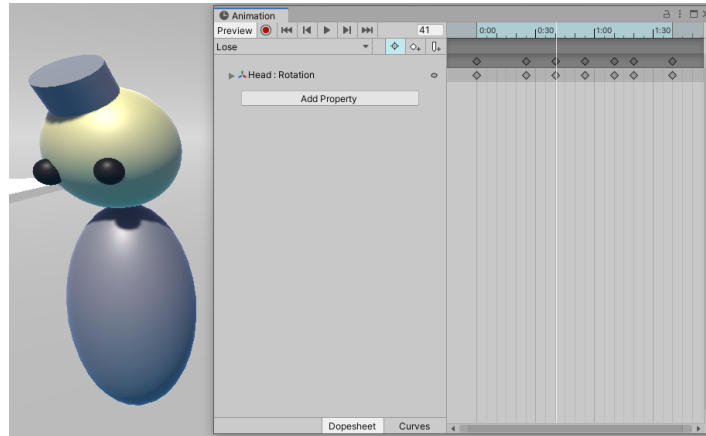


Figure 4.15: Creation of Lose Animation in Unity

#### 4.1.6 Agent Information UI

Additionally in the visualised simulation environment a user may click on an agent to bring up information about them and their parameters (figure 4.16). The information shown includes the voter ID, whether they are a Guru or Delegator, their competence levels, their vote history, their power, their delegate and Guru (which may be themselves), and who each of their Delegators are. Figure 4.16 circles which agent has been clicked. You can see they are a Guru with two Delegators giving them a power of 3, but only one of those Delegators is direct.

This menu can be closed with the close button or the user can go straight to another agent by clicking on it. A user may choose analysis mode to give them more time to view the information in this panel. It makes delegations much clearer in large networks with many delegation lines. The menu can be kept open on a single agent through the simulation to see the stats change in continuous learning runs where the same agents

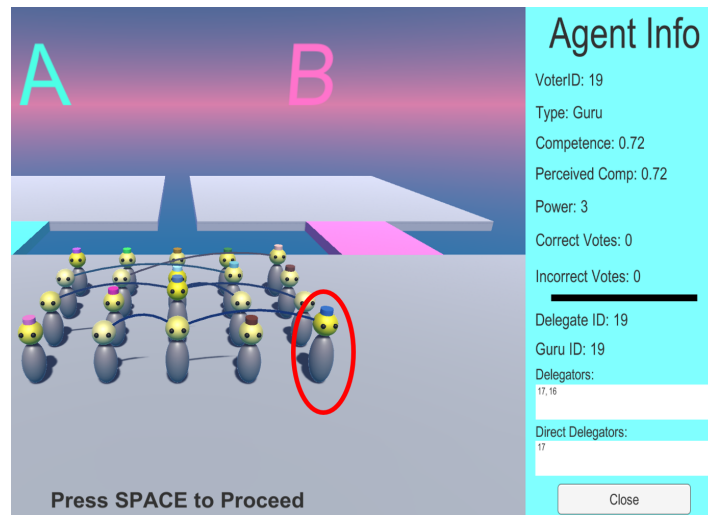


Figure 4.16: Agent Information UI

remain through the simulation. In the case of one shot (which is likely the case in figure 4.16 as the competence and perceived competence are the same) the menu will close after a vote is complete as the agent will be deleted and replaced.

## 4.2 Console Simulation

As well as having the visualised simulations implemented above, the created software also includes the functionality to run simulations without visualising it all in Unity. The purpose of this functionality is to act more like a program running in a console, like many of the models that were run for experiments in previous literature, as running without visualisation makes the program run much quicker, so it is helpful in collecting results of a large number of votes. If a user just wants to see how well a LD system performs overall without needing to see how the individual agents operate they can just run the simulations in console mode.

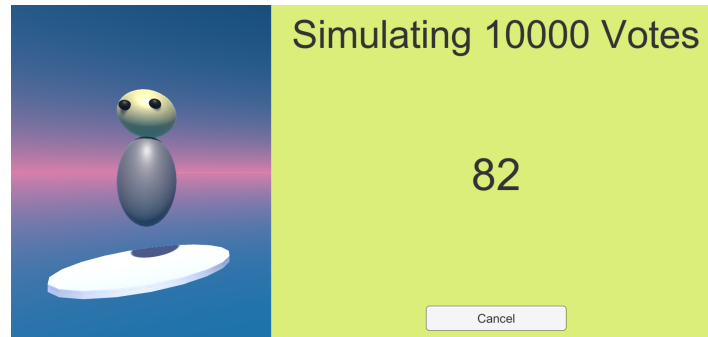


Figure 4.17: Non-Visualised (Console) Simulation UI

The console simulation has a basic UI (figure 4.17), displaying how many votes are being run and the progress so far, with a button to cancel the simulation. The console simulation UI is in the same scene as the start menu (see figure 4.20), simply replacing what is on the screen. Alternative scripts were coded in C# which perform all the same functionality as their visualisation counterparts, but without any of the Unity rendering, “NonSimulationScript.cs” replaces “LiquidDemocracy.cs”, and “VoterScript.cs” replaces “VoterHandler.cs”. There is no separate code for delegations as the delegation handler code was purely for Unity rendering, so delegations are handled with numbers in the voter script.

### 4.3 Simulation Processes

This subsection covers the more technical details of how agents are created and how decisions are made by said agents. This applies exactly the same to both the visualisation and console (non-visualised) simulations, though visualisation figures may be

referred to for clarity. The processes of the simulation are covered in the order they happen after a simulation is started. Most of the same applies to both the One Shot and Continuous Learning simulations but differences will be highlighted.

### 4.3.1 Creating Voter Agents

The first process that happens in every simulation is the creation of the voter agents that will act within the LD voting simulation. The creation of instances of the voter objects is handled by the LD environment. This will be done by either the “LiquidDemocracy.cs” script in the visualised versions creating voter models with “VoterHandler.cs” scripts or by the “NonSimulationScript.cs” script in the console version creating empty voter objects with the “VoterScript.cs” scripts.

When each agent is created they are given a competence of 0.5 plus a value randomly generated by a gamma distribution function. The gamma distribution is a two-parameter family of continuous probability distributions from probability theory and statistics, in which lower values are much more likely to be generated by the PDF than higher values, simulating lots of voters who are not experts in the voting topic with relatively few who are experts and will be very competent.

With gamma distributions there are two different commonly used parameterisations, either with a shape parameter  $k$  and a scale parameter  $\theta$ , or the shape parameter  $k$  and an inverse scale parameter  $1/\theta$ , where the latter are the parameters used by the Gamma function provided by C# library `MathNet.Numerics.Distributions`. The expected value

of  $\text{Gamma}(k, \frac{1}{\theta})$  is:

$$E[X] = k\theta \quad (4.1)$$

In the LD simulation  $k$  is always set to equal 1, meaning the average value generated by the Gamma distribution is equal to  $\theta$ . The competence of any agent A is generated with the following equation:

$$\text{COMP}_A = \min(0.5 + (\text{Gamma}(1, \frac{1}{\theta})/100), 0.95) \quad (4.2)$$

It is then rounded to two decimal places.

So, the competence of an agent A ( $\text{COMP}_A$ ) is on average going to be equal to 0.5 plus  $\theta/100$ . For example in the case  $\theta$  is set to 4, then the average competence of an agent will be 0.54, with the majority of the voters not being experts with competence below the average who are at best only just better than guessing which option to pick each vote, with a few very competent agents getting to higher values such as 0.6 and higher making educated votes and bringing the average up to 0.54. As equation 4.2 shows with the min function, the competence is maxed at 0.95 (with lower  $\theta$  it is rare that values will ever get that high), so no agent can ever have perfect knowledge.

While competence is how likely an agent is to make a right decision, the perceived competence of an agent is how competent other agents think the agent is based on past experience. In One Shot simulations perceived competence is set to be equal to the actual competence of agents, as they are only used once and meant to simulate a

community that has been using LD for a while. In Continuous Learning simulations with learning agents all of the agents start with a perceived competence of 0.5 as it is simulating the LD system being used for the first time where it is assumed agents have no past experiences.

### 4.3.2 Guru or Delegator

After all the agents have been created but before the first vote each agent has to decide if they want to be a Guru or a Delegator. The higher the competence of an agent the more they trust themselves to vote on the topic and so the less likely they are to delegate. If an agent has a competence of 0.5 it will delegate with a 0.95 chance, else the chance of an agent A, with competence  $COMP_A$  of 0.51 or higher, delegating their vote is calculated by the following equation:

$$Delegator_A = 0.95 - \left( \frac{\log(COMP_A * 100 - 50)}{\log(45)} * 90 \right) / 100 \quad (4.3)$$

Which at lowest with a competence of 0.51 equals 0.95, so even the least competent voters may still vote for themselves, and at highest with a competence of 0.95 equals 0.05, so even the most competent voters can delegate if they perceive another voter to have a higher competence, even though it is not possible for true competence to be above 0.95 in this system.

The use of logarithms in equation 4.3 is to have the chance of delegation initially



reduce rapidly as competence increases from 0.5 and have the rate of decrease get increasingly slower when getting into the higher competences. This is to simulate agents becoming much more confident in voting for themselves when having even a little bit of knowledge about a topic. For example the chance of delegating is already under 0.5 with a competence of 0.6, but it then takes much more competence to significantly reduce the chance of delegating anymore, separating the agents that know a lot from the true experts.

Note: The most competent agent in the system and agents that don't know any agents that they see as more competent than themselves may at first choose to be a Delegator, but then upon realising there are no valid delegation options it will choose to change to a Guru.

### 4.3.3 Delegation Decision

All the agents that chose to be a Delegator then have to pick which other agent to delegate to. The LD environment keeps a sorted list of all the agents in the community ordered by their perceived competence. A Delegator can delegate their vote to any agent that comes before them in that list under two conditions: first they must personally know of that agent, meaning it is in their local network, and secondly it must satisfy the following equation:

$$PCOMP_B \geq COMP_A + \alpha \quad (4.4)$$

Meaning an agent A can delegate to another agent B if the perceived competence of agent B ( $PCOMP_B$ ) is higher than the competence of agent A ( $COMP_A$ ) plus the

value of  $\alpha$ , where  $\alpha$  is a globally shared value (it is the same for all agents) that represents how much more competent other agents have to appear for an agent to consider them an acceptable delegation option.

Note: As stated PCOMP (perceived competence) is always equal to COMP (competence) in One Shot simulations, but in Continuous Learning simulations it will start off equal to 0.5 and for the following rounds be calculated with equation 4.9 that is discussed later.

Local networks are explained in the design section and visually represented in figure 3.5, but in short represent the other agents voter in the system is aware of. Each agent has local network represented by a dictionary in C# where the key is the voterID of the agent it knows and the value is the voter object itself. In the case a Delegator is the most competent in its local network it will choose to change to a Guru.

In the case a Delegator only has one acceptable delegation option it will pick it by default, but in the case of multiple options it must choose which one to delegate to. Which agent to delegate to is chosen using weighted chances, with the weight for each potential delegation being calculated based on agent preferences. As discussed in the design section the preferences an agent has are “Preferential Attachment” which is how important the popularity of a potential delegate is, and “Competence Importance” which is how important how much more competent a potential delegate is.

So, a potential delegate B has influence towards being chosen by a Delegator A based on how many Delegators ( $\#Delegators_B$ ) B already has:

$$prefPower_{B \rightarrow A} = (1 + \#Delegators_B)^{PA} \quad (4.5)$$

And it has influence towards being chosen by a Delegator A based on how much its perceived competence ( $PCOMP_B$ ) is above the competence of A ( $COMP_A$ ) plus how much higher perceived competence needs to be to be considered ( $\alpha$ ):

$$compPower_{B \rightarrow A} = (1 + (100 * PCOMP_B - (COMP_A + \alpha)))^{CI} \quad (4.6)$$

The values PA (preferential attachment) and CI (Competence Importance) are numbers (floats) that represent how important each of these values are to the agents. They are set in the main menu and assumed to be the same for all agents in a community. If one is set to 1.0 it is of average importance, 0.5 and 2.0 represent low and high importance. If both PA and CI are set to 0 they are not values held by the agents and every single potential delegate would have an equal chance of being chosen.

The total weight of agent B when being considered as the delegation option for Delegator agent A is the combination of the previous two calculated values:

$$Weight_{B \rightarrow A} = prefPower_{B \rightarrow A} + compPower_{B \rightarrow A} \quad (4.7)$$

And the chance of A choosing to delegate to B is the weight of B divided by the total

weights of all the agents that A has to choose from:

$$Delegates_{A \rightarrow B} = \frac{Weight_{B \rightarrow A}}{TotalWeights} \quad (4.8)$$

A then picks one of the options based on their weighted chances.

#### 4.3.4 Voting and Ending the Round

After each agent has chosen to be a Guru or Delegator, and all the Delegators have chosen who to delegate to and been assigned their Guru, either directly or through transitive delegations, the Gurus make their votes. The voting is extremely simple, each Guru votes for one of two options, the competence level of the Guru is the chance it picks the correct option (the option with better outcomes for the community).

For each Guru the C# Random library is used to generate a random whole number (Integer) between 1 and 100, the competence of the Guru is multiplied by 100 and if it is greater than or equal to the randomly generated number then the Guru votes for the correct option, else it votes for the incorrect option. The C# Random library is what is used for all random number generation for all random choices, including the weighted random choices (Guru or Delegator, who to delegate to, voting, and chance of removing delegation (discussed later)). The Guru votes with the power of 1 plus the number of Delegators they have.

The number of correct decisions by the community is incremented if the correct

option wins the vote, else the number of incorrect decisions is incremented. The stats saved by the Stats Manager for each voting round include the number of votes for each option, the number of Gurus, the average competence behind each vote, and the power of the highest powered Guru. The round in the simulation has finished. If it is the last round then the simulation goes to the statistics screen, else the next round is set up.

In One Shot simulations all the agents are deleted and the process is repeated from the start spawning in a whole new set of voters. Every single round has a different set of voters that go through each of the processes explained so far and then they are deleted and replaced until the desired number of rounds have been completed.

However, in Continuous Learning simulations the agents that were spawned at the start of the simulation are the ones used throughout every single round, and they are learning agents as they learn more about each other as they go on. Instead of just repeating all the processes exactly how they happen in the first round the learning agents have different processes and more decisions to make.

#### **4.3.5 Learning Agents: Assessing Previous Vote and Delegations**

In Continuous Learning as the same agents are used every vote then after each vote the agents will learn more about each other and want to assess what the outcome was and the decisions they made.

After every vote the perceived competence of every Guru who voted in that vote will be updated using the following equation:

$$PCOMP_A = \max\left(\frac{\#Correct_A}{\#Correct_A + \#Incorrect_A}, 0.5\right) \quad (4.9)$$

Simply equating the perceived competence of each Guru as the number of correct decisions they made divided by the total number of votes they have voted in (e.g. if they had voted in 20 votes and got 15 right their perceived competence would be 0.75). However note the max function which means the perceived competence of an agent can never be under 0.5, so even if they have made more wrong votes their perceived competence will still be 0.5, as it is assumed that no agents are actively voting against the interests of the community, so at worst they are guessing.

Agents who chose to remain as Delegators throughout the whole simulation will never update their perceived competence, so it will always remain as the initially set 0.5 value. After many rounds of voting (i.e. thousands of rounds), the perceived competence of the agents that choose to be Gurus (more commonly the higher competence ones) should be close to their true competence.

An example of perceived competence in a continuous learning simulation updating in a visualised simulation is shown in figure 4.18. On the top left it shows all the agents voting with the same colour head representing them all having 0.5 perceived competence in the first round. Then on the bottom left the circled agent and all the other agents who chose the same option were clearly correct as their perceived competence is



Figure 4.18: Learning Agents with Updating Perceived Competence between Votes

updated to 1 (see the right of figure 4.18 for the agent parameters updating after the correct option was revealed) while all the ones who chose the wrong option remain the same colour keeping a 0.5 perceived competence.

Figure 4.18 also represents how it can take a while for correct competences of other agents to be perceived, as the circled agent has a competence of 0.51 but after the first round is perceived to be highly competent as it has only ever been correct.

Every Guru who voted correctly in the last vote will remain a Guru (remember in this model their votes are not split into topics). The Gurus who were wrong however will remake the Guru or Delegator decision, with the chance of being a Delegator always being the same (using the value calculated in equation 4.3). This should on average result in the higher competence voters mostly remaining as Gurus, where lower com-

petence voters who chose to be Gurus should eventually end up being Delegators most of the time.

As for the Delegators from the previous vote, the ones whose delegation ended up with them having a Guru (through direct or transitive delegation) who made a correct decision will keep their delegation the same for the next vote. The Delegators who had a Guru that voted for the incorrect decision will have a chance to remove their delegation and either be a Guru or pick a different delegation option.

First each direct Delegator of an incorrect Guru will get to pick to keep or remove their delegation, if they choose to remove their delegation then all their Delegators (if any) will keep their delegation to them, however if they keep their delegation to the incorrect Guru then their Delegators will also have the chance to remove their delegation to not have that Guru. This chance to remove a delegation that ends up with an incorrect Guru continues down the delegation chain from direct Delegators until a Delegator in the chain has removed the delegation (and keeps all of its Delegators) or the last Delegator in the chain is reached).

When any Delegator agent A gets the chance to remove a delegation to stop having an agent B as Guru (either from direct or transitive delegation) the chance is equal to 50 percent minus how much more competent the Guru B is perceived to be than the



Delegator A is, in equation form:

$$RemoveDelegation_{A \rightarrow B} = (50 - 100 * (PCOMP_B - COMP_A))/100 \quad (4.10)$$

The agents who remove their delegation from the previous vote must then make the Guru or Delegator decision again with the same chance calculated in equation 4.3. If an agent who has just removed a delegation chooses to be a Delegator again they will only make a delegation that results in them having a different Guru, if there are no acceptable options they will choose to be a Guru. An agent can go back to having a Guru it left after the next round of voting; only agents that were removed as Guru in the previous round cannot be picked again.

In visualised continuous learning simulations as the same set of voters are used instead of them being spawned at the start again they will be shown moving back from the voter area to their start positions as shown in figure 4.19, keeping their body colour from their previous vote option so the user can tell who was right and wrong. When the agents are back in their start positions this is where who are Gurus and who are Delegators will change along with and who delegates to who.

After all the agents have either kept their previous decisions or changed using the chances calculated above they vote again with the new delegation decisions. After the initial processes described above these processes of voting and the assessing and changing being a Guru or Delegator and who to delegate to continues until the desired number of rounds have been completed, at which point the simulation ends.

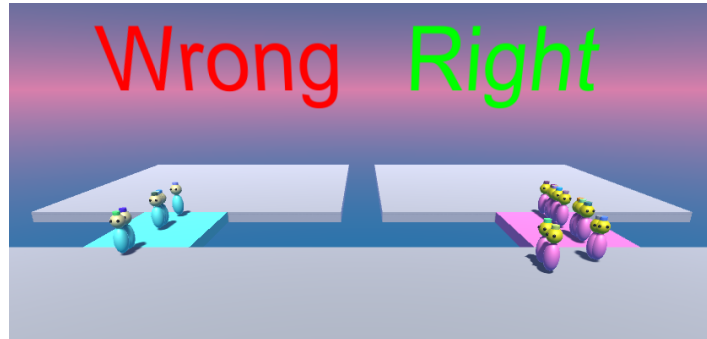


Figure 4.19: Learning Voter Agents Moving Back to Start for Next Vote

The idea behind these continuous learning simulations is that they should show the benefit of a single community using a liquid democracy voting system, getting more correct votes than direct voting would, and the average vote competence rising as the system is used and good delegations are made, as shown to be the case in the stats recorded to a CSV file in figure 4.22, with the number under the “Avg Vote Comp” column rising from start to finish.

## 4.4 Simulation UI and Statistics

This subsection briefly covers some of the UIs created as part of the Liquid Democracy simulation software as well as the statistics that are collected and how they are stored for the user.

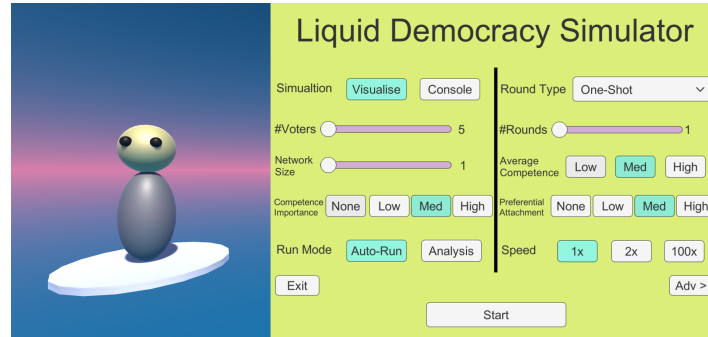


Figure 4.20: Menu Screen of Liquid Democracy Simulation created in Unity

#### 4.4.1 Menu Screen: Setting Variable Values

The Liquid Democracy program is built from Unity into an executable (.exe) file that can be run by users. The first thing the user will see when running the program is the main menu (figure 4.20) where they can choose the settings they want and run a simulation. The main menu allows users to set the simulation to be displayed with “Visualise” or run in the background “Console”, and have the drop down to pick between One Shot and Learning simulations. The other basic settings include the number of voters and rounds, the local network size, and for visualised simulations whether to auto run or run in analysis mode, and the speed of the run.

Looking at the more complex options, for the “Average Competence” setting “Low” makes the average competence of agents 0.54, “Med” is 0.58 and “High” is 0.62 (very little to no mistakes would be expected in “High”). For “Competence Importance” the value of CI in equation 4.6 is set to a different value depending on the button highlighted (by being clicked) exact values are “None”: 0, “Low” 0.5, “Med”: 1, “High”: 2. For “Preferential Attachment” the value of PA in equation 4.5 is set to the same values

as CI is for the equivalent buttons.

The Unity SceneManager is used to handle different scenes in the program, the menu scene is loaded first and is also the scene where the console simulation runs, by pressing the start button with “Visualise” set on the simulation scene is loaded. At the end of a simulation (visualised or not) the statistics scene is loaded.

As the project progressed more experiments were developed with new parameters, so a secondary settings screen has to be added, shown in appendix A, figure A.1. It is navigated to and from using the clearly labelled buttons. The space left on this screen leaves lots of room for extensions and implementing models from previous work.

#### **4.4.2 Statistics Screen and Manager**

The Stats Manager as discussed earlier is created in the menu scene and has a script called “StatsManager.cs”. It manages the menu UI and has the important job of carrying the setting set by the user through to the simulation. As well as managing settings it records the statistics from the simulation (hence the name), with the most basic of those being the number of correct and incorrect decisions made (result decisions not each individual voter decision).

Also stored for each round is the number of Gurus per vote, to get the average overall for the simulation (the average number of Delegators can be got for this), as well as the power of the highest powered Guru to get the average highest power. This

highest power average can be used to tell if a community with certain settings is prone to having super voters which can be detrimental to their performance.

The final stat to be collected for each round is the average vote (not voter) competence. For direct voting the average vote competence would be the sum of all the voter competences divided by the number of voters, however in LD it is the sum of all the Guru competences multiplied by their powers divided by the number of voters. This is because if a voter has a Guru their competence for the vote is replaced by the competence of their Guru. By using LD and having all agents that delegate have Gurus with higher competence you would see the average competence increase, the average of the average competence is calculated for the overall simulation.

The simulation scene displays the simulation statistics (figure 4.21). It is managed by a script called “StatsUIHandler.cs” which does the work of calculating the averages of the stats collected by the stats manager and writing them to the UI. The stats UI also displays some of the settings that the user set for recording purposes, and it also has a button to go back to the main menu (by loading the menu scene).

### 4.4.3 Exporting Statistics to a CSV

As well as the statistics being presented to the user through the UI, it is also written to a .csv file and stored locally on the machine running the program, implemented using the C# System.IO library and StreamWriter object.

Figure 4.22 shows an example of one of the CSV files (openable in software such

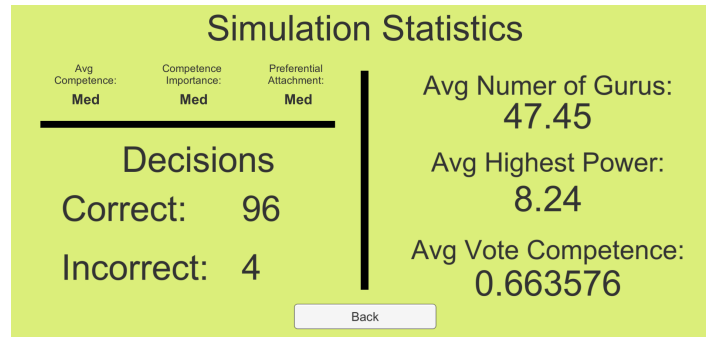


Figure 4.21: Statistics Screen Showing Results and Stats from run Simulation

Run	Correct Votes	Incorrect Votes	#Gurus	Highest Power	Avg Vote Comp
1	57	43	100	1	0.569120572
2	52	48	78	3	0.577099057
3	59	41	64	5	0.584850802
4	44	56	49	7	0.596221141
5	65	35	37	7	0.606963699
6	63	37	34	15	0.623115116
7	83	17	40	13	0.622472741
8	53	47	36	13	0.630308145
9	63	37	30	13	0.632550435
10	71	29	28	12	0.639934983
11	88	12	23	12	0.644988506
12	74	26	22	12	0.642443999
13	52	48	21	12	0.642200055
14	59	41	22	15	0.630188343
15	70	30	19	14	0.638236842
Summary					
Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp	
14	1	40.2	10.26667	0.618712963	

Figure 4.22: CSV created by Simulation Software, showing individual round and overall summary stats

as Microsoft Excel and Google Sheets). The created CSV contains a summary of the results at the bottom like the UI does, but additionally it also stores the statistics from each individual round listed one by one above the summary (available for all settings including both One Shot and Continuous Learning simulations), including the number of correct and incorrect individual votes for making each decision.

# Chapter 5

## Experimental Design

The created liquid democracy simulation software needs to be put through a study to explore what it shows about the performance of a liquid democracy voting system under different conditions. The study should ideally show the aim of the system was met, which was to provide users who are thinking of implementing such a voting system in real life with information about why it is in fact beneficial to do so, also highlighting how such a system could fail and what to do to avoid this.

This subsection covers the details of design of the planned studies, looking at the individual aims of the different experiments, the independent and dependent variables, and the processes of setting up the different experiments to get results from the study. Then the next section will look at the results from the study described here.

## 5.1 Aims and Scope

The main aim of the study is simple: to produce data that allows for the evaluation and discussion of liquid democracy voting systems. The data should show how the created liquid democracy model in the simulation system performs under a range of different conditions by changing different independent variables, which allows for the discussion of the best possible implementation of the given system. This study is focused mainly on discussion and development of ideas about liquid democracy, rather than proving set hypotheses.

While the main aim is to produce data that allows for discussion of the liquid democracy system in general, there are some smaller individual aims that can be highlighted.

### **General Study Aims:**

- **A1** - Compare Liquid Democracy to Direct Voting in One Shot voting
- **A2** - Measure the performance of the LD system (by looking at change in Dependent Variables) under changing voter values such as preferential attachment and competence importance
- **A3** - Measure the the performance of the LD system with different local network sizes (the number of voters each voter knows)
- **A4** - Measure the performance of the LD system with different alpha values (the minimum amount the perceived competence of an agent has to be greater than the competence of a Delegator for the Delegator to consider delegating to that agent)



- **A5** - Compare Liquid Democracy to Direct Voting in communities of learning agents
- **A6** - Compare Liquid Democracies with different alphas in communities of learning agents

A big part of the contribution provided by the creation of the simulation system is that it is meant to provide information for users to help them understand liquid democracy, so there is some set information that is aimed to be provided.

**User Information Aims:**

- **A7** - Show that Liquid Democracy can perform better than Direct Voting
- **A8** - Show that Liquid Democracy voting increases the average voting competence of a community
- **A9** - Show the danger of Super Voters with a large voting power

Of course, even though the main focus of the study is to produce discussion, not to proving hypothesis, there are some hypotheses that can be made about the performance of the system that would be expected to be correct if the system has successfully modeled liquid democracy based on previous literature covered in the literature review.

**Hypotheses:**

- **H1** - A well configured Liquid Democracy system will perform better than (make fewer incorrect decisions than) Direct Voting

- **H2** - Learning agent communities with well configured Liquid Democracy voting systems will on average make fewer incorrect decisions than learning agent communities using Direct Voting
- **H3** - Increasing the number of voters each voter is aware of (network size) will increase the average vote competence of the community
- **H4** - Increasing the number of voters each voter is aware of (network size) will increase the average power of the highest powered Guru
- **H5** - Liquid Democracy systems with very large network sizes will perform worse than those with smaller network sizes (super voter problem)
- **H6** - Increasing Preferential Attachment for Gurus will increase the average power of the highest powered Guru
- **H7** - Increasing Preferential Attachment for Gurus will make Liquid Democracy systems perform worse (super voter problem)
- **H8** - Increasing Competence Importance for Gurus will increase the average vote competence of the community
- **H9** - Increasing Competence Importance for Gurus will make Liquid Democracy systems perform better

How the success of these aims and the correctness of these hypotheses will be analysed will be described through the following subsections.

## 5.2 Dependent Variables

The system records a number of statistics (see figure 4.21) which will be recorded as dependent variables as the different independent variables of each experiment are changed. The dependent variables that will be recorded are:

- **Incorrect Decisions** - This is the number of times there were more incorrect than correct votes for a decision. This will be the primary measure of performance of the community with fewer incorrect decisions indicating better performance.
- **Correct Decisions** - The number of correct choices made by the community (equal to total decisions minus the incorrect decisions). Also a measure of performance, but incorrect will be used primarily in graphs and analysis.
- **Average Vote Competence** - The sum of the competence of each Guru in a vote multiplied by their power, all divided by the total number of Gurus, all divided by the total number of decisions, showing the average competence behind all the decisions.
- **Average Number of Gurus** - The sum of the number of Gurus in each round divided by the total number of rounds, showing on average how many Gurus there are voting in the system (and trivially how many Delegators).
- **Average Highest Power** - The sum of the powers of the highest powered Guru each round divided by the total number of rounds, showing on average how powerful the most delegated to Guru is in the system (high values may indicate existence of super voters).

### 5.3 System Study Setup

The study is ran by simply running the simulation with changing independent variables.

The same basic steps to do a run for one of the experiments are as follows:

1. Load up simulation software to main menu (figure 4.20) (or go back to menu from end of last run).
2. Set simulation to "Console" for quicker run.
3. Set number of voters to 100
4. Set number of rounds to 10,000
5. Set mode to "One Shot" or "Learning" depending on type of experiment
6. Set relevant independent variable of the experiment to desired value
7. Set other values (control variables) to set values for given experiment
8. Start the simulation, wait until all 10,000 rounds of voting are completed
9. Take notes of stats on statistics screen or use created CSV
10. Move on to next run for the experiment or close program if finished

100 voters are used for every experiment for a middle ground between community size and run time. 100 voters gives results for a medium sized community. While a lot of communities (i.e. countries using liquid democracy for political votes) will be bigger

than this, the run time for larger numbers of voters (e.g. 1000) is simply too long to be able to collect enough data with.

The average competence will by default be set to "Med", this sets the average competence of agents to 0.58, with the vast majority being lower and close to 0.5 and a few being much higher to represent the experts (simulated using a gamma distribution). This should work well for 100 voters as there will on average be few experts; however if there were a larger number of voters (e.g. 1000) having the average of the gamma distribution be 0.58 may produce too many experts than would be realistic hindering validity of the results. Time allowed some experiments to be ran with average competence set to "Low" (0.54) to see if less competent communities would perform differently with the same systems.

## 5.4 One Shot Studies

One shot studies are set up using the outline described above, with the mode being set to "One Shot" in step 5.

10,000 votes are run to give a good measure of the percentage of incorrect decisions a community using Liquid Democracy with the set variables would make. One Shot simulations use a different set of voters each round with all perceived competences equal to true competences so 10,000 should provide a good estimate of performance. The creation of 100 new voters each round slows the simulation so more rounds would

not be feasible in the scope of this project.

#### 5.4.1 Independent Variables: Preferential Attachment and Competence Importance

The Preferential Attachment and Competence Importance parameters of the agents in the system will be varied to measure their effect on the system as a whole (Aim **A2**). They will be set on the menu screen to the 4 different options of None, Low, Med, High, which set the values of PA in equation 4.5 and CI in equation 4.6 to 0, 0.5, 1, and 2 respectively.

Increasing preferential attachment increases how important popularity of potential delegates is to agents. Hypothesis **H6** predicts increasing the preferential attachment will increase the highest power and **H7** predicts that this will make the community perform worse. This may contribute to aim **A9** if it does in fact create super voters and cause more incorrect decisions.

Increasing competence importance increases how important how much more competent potential delegates are to an agent than the agent itself. Hypothesis **H8** predicts increasing competence importance will increase the average competence of the community, and **H9** predicts it will make the community perform better.

While increasing preferential attachment from None through to High competence importance will be set to None, and likewise as competence importance is varied prefer-

ential attachment will be set to None. These experiments will be run for network sizes of 10 and 100, and the alpha value will be set to 0.02.

The predictions for this study summarised to a basic level are that prioritising competence is good and preferential attachment is bad for liquid democracy systems.

### 5.4.2 Independent Variable: Network Size

The local network size of each agent will be varied to see how it effects the dependent variables of the simulation statistics (aim **A3**). The network size will be set to values ranging from 1 to 100 (at 100 all voters are aware of each other).

When the network size is 1 this will be simulating direct voting, so comparing this to higher network sizes of 2 and higher will be comparing Direct Voting to Liquid Democracy (Aims **A1**, **A7**, **A8**). If hypothesis **H1** is correct then there will be network size(s) greater than 1 that perform better than a network size of 1.

The higher network sizes increase the chance of agents having more potential options of agents to delegate to, therefore more agents may delegate to higher competence agents, increasing the competence of votes (hypothesis **H3**). But this also means Gurus may have more Delegators which would increase their power (hypothesis **H4**); this could lead to super voters which may make the system perform worse (hypothesis **H5**, aim **A9**).

For this both preferential attachment and competence importance will be set to Med, assuming they are both valued by agents, and they are valued equally. The alpha value will again be set to 0.02.

### 5.4.3 Independent Variable: alpha

The alpha value in the previous experiments was set to 0.02. The alpha value represents how much more competent other agents have to be perceived than a Delegator for that Delegator to see them as acceptable to delegate to ( $\alpha$  in equation 4.4).

This has been assumed to be 0.02 (i.e. 2% higher), as 0.01 is assumed to be too close for agents to tell the difference, and then anything higher would be too high as there is no way to enforce this in real life as the point of liquid democracy is that anyone can delegate to anyone, and some agents may not know of any other agents will perceived competence high enough to delegate to. So, the assumption that agents only delegate to higher perceived agents is already restrictive enough without restricting them to only delegating to agents that are much higher.

However, there are ways to more clearly present high competence voters, e.g. Guru advertisements, which may direct more voters to only delegate to voters who have proven they are experts. So varying the alpha value to higher values simulates those communities who have been well provided with information about not wasting their delegation and good Gurus. If these higher alpha simulations perform better it provides more information to users, telling them the importance of clearly giving voters good



delegation options. The limiting factor of these experiments is some low competence agents may not know of any agents with high enough perceived competence to delegate to so not delegate at all which may not be realistic.

The alpha value will be set to 0.02, 0.05, 0.1, and 0.2, requiring agents to be 2%, 5%, 10%, and 20% higher to delegate to them respectively (aim **A4**). These tests will be run across the same varying network sizes as the experiment before (1 to 100), and again preferential attachment and competence importance will be set to Med.

## 5.5 Learning Agents Studies

Continuous learning agent studies are set up using the outline described above, with the mode being set to "Learning" in step 5.

In these experiments Preferential Attachment and Competence Importance will be set to Med as before.

Unlike in one shot only one set of agents is used in each simulation, because of this each simulation experiment is repeated 10 times, still with 10,000 voting rounds each, so averages can be calculated. Running these tests multiple times and taking an average is required as if, like in one shot, only one simulation was run the one set of agents that were produced might have a higher or lower than average competence, making results seem better or worse than they would be on average.

These learning simulations are around twice as fast as the one shot counterparts as they don't have to create 100 new agents every round, so repeating 10 times will not be too time consuming, but doing much more than that to get a more accurate average would take too long.

### 5.5.1 Direct Voting vs Liquid Democracy

Communities of 100 voters using Direct Voting will be compared to communities of 100 voters using Liquid Democracy (aim **A5**). For Direct Voting the network size will be set to 1 so voters all have to be Gurus and vote directly. That is, even though they are "learning" they will never find other agents as they don't have a network to make transitive connections through. For Liquid Democracy the network size will be set to 10, meaning all voters start off knowing 9 other voters. Because they are learning agents if they are ever given a Guru through transitive delegations that they don't know they will add it to their local network, so network sizes of voters can grow.

In Direct Voting agents have to vote directly so the average vote competence will remain the same, whereas in Liquid Democracy they can delegate to higher perceived competence agents so the average vote competence should rise (aim **A8**). As perceived competences are not always correct it will take a while for good delegations to be made, but it is assumed (hypothesis **H2**) under the right conditions the higher average vote competence allowed by Liquid Democracy will make it perform better than Direct Voting (aims **A7,A8**).

A potential issue that may make Liquid Democracy perform not so well, potentially worse than direct voting, is the growing network size creating Gurus with really high voting powers (super voters) that swing decisions in the wrong direction (hypotheses **H4**, **H5**). If Liquid Democracy performs well, it will show users a good reason to use it alongside it increasing the average vote competence. On the other hand, if it does have issues caused by the growing network sizes it will show them the issue of super voters (aim **A9**) that should be avoided, so the data will be helpful either way.

For these experiments alpha will be set to 0.02, at least to start with.

### 5.5.2 Independent Variable: alpha

Changing the alpha value in the One Shot experiments found to have a significant effect on the results of the simulations, so it was changed for Continuous Learning Agent simulations too (aim **A6**).

Increasing this alpha value could lead to better performance for a community, as there will be less delegations to mid range competence voters, removing the chance of them becoming super voters. However, it could also lead to the community performing worse, as limiting to just delegating to very high competence voters could lead to the high competence voters being super voters as they are the only option, or many voters may not have any acceptable delegations so they will always have to vote directly and never learn about good Gurus transitively (of course in real life there would be other ways to learn about Gurus, e.g. online advertisements).

# Chapter 6

## System Study Results and Discussion

The results of the experiments from the study are given in this section along with discussion about them.

Although the results of the experiments are not time based, the built software does require graphical power and takes a while to run simulations, so the specifications of the PC used to run the experiment simulations are given here:

- **Operating System** - Windows 10 Home
- **Graphics Card** - NVIDIA GeForce GTX 1050 Ti
- **CPU** - Intel Core i7-3770 3.40GHz
- **RAM** - 16GB

100 Voters; Local Network Size: 10					
Preferential Attachment	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
None	9578	422	46.971	7.8789	0.65583
Low	9531	469	46.926	8.0423	0.655931
Med	9477	523	46.9159	8.3605	0.656365
High	9496	504	47.0028	8.8536	0.657039

Table 6.1: One Shot Results: Average LD Community Statistics by Preferential Attachment Level for Network Size of 10

For the console (non-visualised) simulation experiments this gave average running times of:

- **For 10,000 Rounds of One Shot** - 1 Hour and 10 Minutes
- **For 10,000 Rounds of Learning** - 30 Minutes

## 6.1 One Shot: Effect of Changing Agent Values

### 6.1.1 Results of Preferential Attachment Experiment

10,000 simulations were run for all four different Preferential Attachment values for a community of 100 voters with both a local network size of 10 (results in table 6.1) where each agent is aware of 10% of the community, and a local network size of 100 (results in table 6.2) where each agent is aware of the whole community.

For a local network size of 10 increasing the preferential attachment from None to Low and Low to Med increases the number of incorrect decisions made (see the incorrect decisions column in table 6.1). Increasing from Med to High actually results in

100 Voters; Local Network Size: 100					
Preferential Attachment	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
None	9479	521	43.9	11.0555	0.663302
Low	9416	584	43.9176	11.8652	0.664143
Med	9283	717	43.9565	14.6217	0.667613
High	8510	1490	43.8705	29.9985	0.682712

Table 6.2: One Shot Results: Average LD Community Statistics by Preferential Attachment Level for Network Size of 100

slightly fewer incorrect decisions made.

For a local network size of 100 as preferential attachment is increased the incorrect decisions increase, by more every time, giving an exponential increase with the biggest difference being between Med and High. With no preferential attachment in this case there are only 521 incorrect decisions, with High there are 1490.

In both cases of network size, 10 and 100, the average highest power rises with the preferential attachment increasing seemingly at a exponential rate (see the relevant columns in both tables). The rate of increase is higher for the network size of 100, overall from None to High rising from 11.06 to just under 30, where in network size of 10 has it rise from 7.88 to 8.85.

The average number of Gurus, again in both cases, remains around the same value with preferential attachment having no effect on this. The average vote competence is barely changed for all values of preferential attachment but does very slowly increase with it in both cases.

100 Voters; Local Network Size: 10					
Competence Importance	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
None	9578	422	46.971	7.8789	0.65583
Low	9616	384	47.009	7.9645	0.659486485
Med	9616	384	46.9966	8.0819	0.66444294
High	9679	321	46.9738	8.4312	0.671875322

Table 6.3: One Shot Results: Average LD Community Statistics by Competence Importance Level for Network Size of 10

100 Voters; Local Network Size: 100					
Competence Importance	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
None	9479	521	43.9	11.0555	0.663302094
Low	9595	405	43.9697	12.0096	0.676084392
Med	9639	361	43.9078	14.0936	0.690754786
High	9565	435	43.9062	19.5223	0.716187323

Table 6.4: One Shot Results: Average LD Community Statistics by Competence Importance Level for Network Size of 100

### 6.1.2 Results of Competence Importance Experiment

10,000 simulations were also run for all four different Competence Importance values for a community of 100 voters with both a local network size of 10 (results in table 6.3) and a local network size of 100 (results in table 6.4).

For a local network size of 10 increasing the competence importance from None to Low decreases the number of incorrect decisions, from Low to Med has the number of incorrect decisions remain the same, and from Med to High again reduces the number. Overall increasing from None to High reduces the number of incorrect decisions by 101.

For a local network size of 100, when compared to 10, the incorrect decisions are higher for all competence importance values except for Med. A competence importance

is increased from None to Low and from Low to Med the number of incorrect decisions decreases, as competence importance is raised from Med to High the number of incorrect decisions increases.

The average highest power in both cases increases at an exponential rate with competence importance, similarly to how it did with preferential attachment but much less of an increase overall. From None to High it increases from 7.88 to 8.43 for network size 10, and from 11.06 to 19.52 for network size 100. Again the average number of Gurus remains around the same value.

The average vote competence in both cases increases with the competence importance at a seemingly linear rate, by a much more significant amount than it did with preferential attachment. It increases by just under 0.02 for a network size of 10, compared to under 0.005 in the preferential attachment counterpart, and by over 0.05 for a network size of 100, compared to under 0.02 for the preferential attachment counterpart.

### 6.1.3 Discussion

The results of these experiments show the effects of changing agent values, satisfying aim **A2**. Increasing the value of preferential attachment increases how much agents value popularity, and increasing competence importance increases how much they value knowledge about the vote.

Increasing preferential attachment increased the average highest power in the votes,



as was predicted by hypothesis **H6**. This prediction was trivial as if the importance of popularity is increased then the agents with the most delegations are most likely to receive new delegations making them more powerful.

It was predicted that because of this increase in highest power caused by popularity that the number of incorrect decisions would rise (hypothesis **H7**). This was the case except for increasing from Med to High in the network size 10 experiment. This was predicted and is mostly the case because voters with high powers can swing votes in the wrong direction (showing the super voter problem, aim **A9**). The reason the incorrect decisions did not go up when increasing from Med to High in a network size of 10 is likely because the network size was too small for there to be any real effect of increasing the value of popularity more, where with a bigger network size like 100 the issue can get much worse with more potential Delegators to one Guru.

Favouring popularity does not help a community on its own as it means agents are possibly not making good delegation to competent Gurus, but instead just ending up with popular Gurus, which can easily be an issue in real life. The average competence does rise a small amount as preferential attachment is increased, the reason for this is likely down to the assumption that Delegators can only delegate to higher competence agents, so when high competence agents delegate to higher competence agents the low competence agents will be more likely to delegate to them too because of their popularity, without this assumption this small increase cannot be assumed.

Increasing competence importance increased the average vote competence in the votes, as was predicted by hypothesis **H8**. This was another trivial assumption as increasing how much Delegators care about competence of course makes them on average delegate to higher competence agents, increasing the average vote competence in the community.

It was predicted that increasing competence importance would cause fewer incorrect decisions to be made (hypothesis **H9**). This was the case overall for a network size of 10 going from competence importance None to High, but for a network size of 100 it was only true going from None to Med, going from Med to High increased the incorrect decisions in this case.

The reason for the incorrect decisions decreasing in the cases where it does is clear, the lower competence agents who are more likely to be wrong are likely to delegate their vote to a much more competent agent who is likely to make the right choice, increasing the number of correct votes. The difference is these cases did not increase the average highest power too much where going from Med to High in a network size of 100 did.

The average highest power increases as competence importance increases because the most competent agents become more powerful. In the network size of 10 this is fine as it never increases much and the benefit of the increases competence outweighs it. This is the same for the network size of 100 except when increasing from Med to High, where the average highest power rises a lot outweighing the benefit of the increased

competence, so there is a high competence high powered voter, but even though it is high competence it can still make mistakes that hinder the community, but this is still better than not valuing competence at all.

These results show it is important for voters to value competence, and valuing popularity can be harmful, valuing both can even out well as the competent voters should be some of the popular ones. They can both potentially cause the issue of super voters which users should take note of (aim **A9**). Note that the average number of Gurus is not affected by either of these things which is to be expected as these values are not factored into the Guru or Delegator decision.

## **6.2 One Shot: Effect of Changing Local Network Sizes**

### **6.2.1 Results of Network Size Experiment on Med Competence Communities**

To measure the effect of network size on the performance of Med competence (average 0.58) communities using Liquid Democracy 10,000 one shot voting rounds were done for a number of different sizes from 1 to 100. The full set of results is shown in table 6.5. The network size of 1 represents Direct Voting, not Liquid Democracy, as there could be no delegations.

The effect of network size on performance (measured by incorrect decisions) is shown

Network Size	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
1	9308	692	100	1	0.579804
2	9516	484	72.8493	3.3217	0.608035
3	9603	397	61.1293	3.9424	0.626068388
4	9594	406	55.404	5.0613	0.636692665
5	9613	387	52.22	5.5962	0.644596
10	9600	400	46.9741	8.2146	0.664146
15	9628	372	45.6169	9.6987	0.672424
20	9629	371	45.0599	10.78	0.677207
25	9586	414	44.7051	11.6065	0.680837
50	9552	448	44.0101	14.3633	0.68759
75	9539	461	44.011	15.3603	0.68981
100	9555	445	43.9008	14.8205	0.691261

Table 6.5: One Shot Results: Average LD Community Statistics by Network Size

on the graph in figure 6.1. The biggest increase in performance is going from network size 1 (Direct Voting) to network size 2 (which is Liquid Democracy as it allows transitive delegations), reducing the incorrect decisions by 208. In fact all cases of liquid democracy (network sizes 2 to 100) outperform direct voting here.

The communities using LD with network sizes between 3 and 20 all perform very well, with just over 400 incorrect decisions or under each. The best performance seems to come from networks sizes between 15 and 20 with the best recorded being only 371 incorrect decisions with a network size of 20 (down 321 from 692 in direct voting).

Going up through the network size values from 20 the communities actually start to perform worse making more incorrect decisions each time, with the exception being the increase from 75 to 100 where it actually performs slightly better.

The average competence of the votes of the community increases as the network size increases (shown in figure 6.2). The increase appears logarithmic with it increasing a

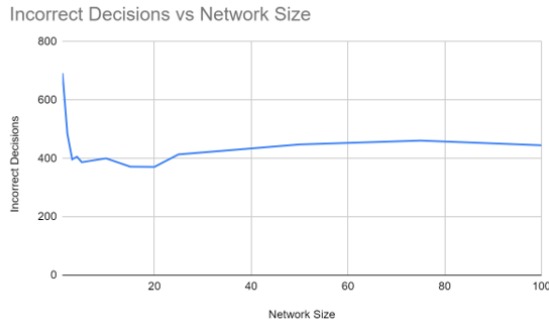


Figure 6.1: Graph of Incorrect Decisions by Network Size

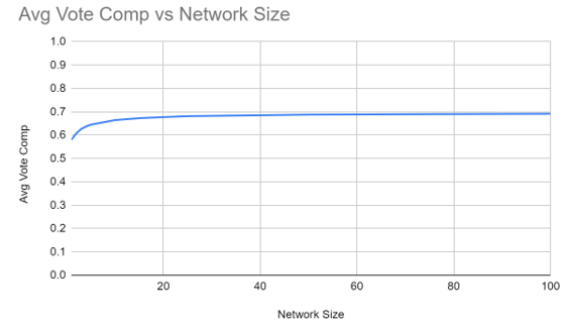


Figure 6.2: Graph of Average Vote Competence by Network Size

lot going from direct voting to a network size of 2 all the way up to 20 where it starts to level out, overall increasing by 0.11 (11% competence) from 0.58 to 0.69.

The effect of network size on the average highest power is shown on the graph in figure 6.3. As the network size increases between 1 and 75 the average highest power increases, but going from a network size of 75 to 100 the average highest power decreases, similar to what happened with the incorrect votes.

The effect on the average number of Gurus is shown on the graph in figure 6.4. The average number of Gurus is logarithmically decreasing as the network size is increased, with the decrease slowing massively after network size 10 and leveling out around size 25 with around 44 Gurus.

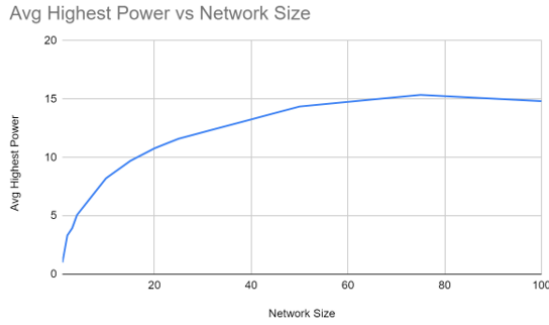


Figure 6.3: Graph of Average Highest Power by Network Size

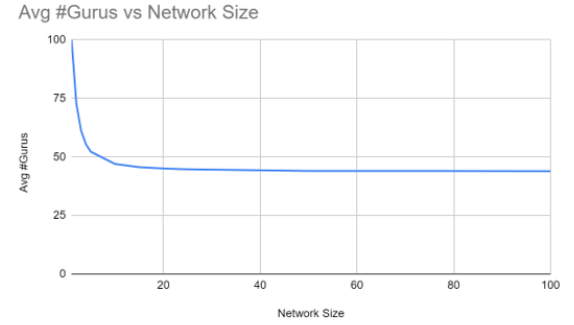


Figure 6.4: Graph of Average #Gurus by Network Size

Network Size	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
1	7632	2368	100	1	0.540039047
2	7907	2093	75.3452	3.2506	0.555413165
5	8022	1978	47.4631	5.9721	0.579328355
10	8053	1947	37.2885	9.8157	0.595112213
15	7938	2062	34.464	12.5284	0.602084942
20	7881	2119	33.0375	14.8454	0.60716274
25	7845	2155	32.4134	16.5928	0.609836181
50	7608	2392	31.0456	23.7168	0.618274941
75	7466	2534	30.7068	27.592	0.622138902
100	7532	2468	30.5129	27.22	0.623978368

Table 6.6: One Shot Results: Average LD Community Statistics by Network Size for Low Competence Communities

## 6.2.2 Results of Network Size Experiment on Low Competence Communities

The network size experiment was repeated for Low competence (average 0.54) communities to see if similar performance at the given network sizes could be expected. The results of 10,000 runs for different network sizes is shown in table 6.6.

The number of incorrect decisions for different network sizes in low competence communities is shown in graph 6.5. Again size 1 represents Direct Voting and increasing

the size to 2 gives the best increase in performance, bringing down the number of incorrect decisions by 275 from 2368 to 2093. Slightly different from the med competence communities, the low competence communities appear to perform best with network sizes between 5 and 10, with the best recorded performance being 1947 incorrect votes for network size 10 (down 421 from direct voting).

There is a big difference with the low competence results, as while the incorrect decisions decrease going up the network size values from 1 to 10, they actually start to increase going up from 10 to 75 meaning the low competence communities do worse with these higher network sizes earlier than the med communities did. Again the incorrect decisions decrease slightly when going from 75 to 100.

The major difference here as visible in graph 6.5 is that the rate the incorrect decisions increases is much faster. This actually results in Liquid Democracy performing worse than Direct Voting in some cases for low competence communities, it is worse for recorded network sizes 50, 75, and 100, with the worst being 75 with 2534 incorrect decisions (166 more than direct voting).

The average vote competence increases as network size increases (see table 6.6) much like how it did for med competence communities, but leveling out at a lower value (which is to be expected with a lower average competence), increasing by about 0.08 (8% competence) from network sizes 1 to 100.

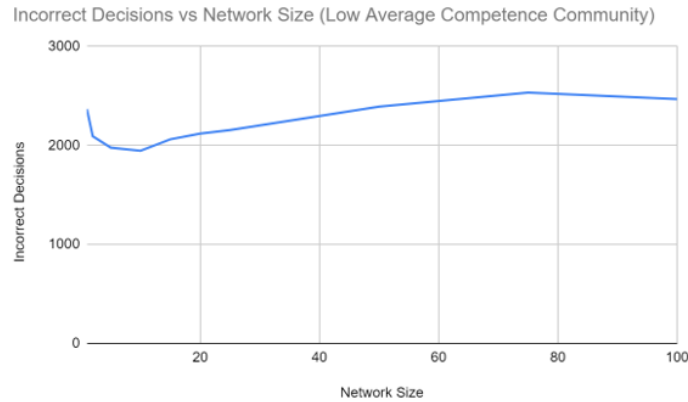


Figure 6.5: Graph of Incorrect Decisions by Network Size (Low Competence)

The average highest power again increases as the network size goes up, again except for when it goes between 75 and 100. However, in the low competence communities it gets to a much higher peak value of 27.59, compared to only 15.36 for med competence communities.

The average number of Gurus again decreases logarithmically, but here it doesn't start to level out until around a 50 plus network size, and the value it levels out to is just below 31 (13 lower than the med case).

### 6.2.3 Discussion

Changing the local network sizes of the agents allowed for comparison between direct voting (network size 1) and liquid democracy (any size greater than 1). For medium competence communities it was found that in these one shot simulations liquid democracy performs better than direct voting for any network size, validating hypothesis **H1**



and meeting aim **A7**. For low competence communities liquid democracy only performed better under low network sizes, meeting the right configuration criteria of **H1**.

The average vote competence increased as network size increased (**H3**, **A8**), which is because bigger network sizes means more delegation options, which means more agents delegating to more competent agents.

The average highest power increased as network size increased (**H4**), except for when going from 75 to 100. The reason for this is bigger network sizes allow for more delegations to be made to a single agent, and with values of high competence and popularity this will of course happen. It fell between 75 and 100 as all agents could see each other so delegations would likely split between a few of the best options.

For the medium size experiments the least incorrect decisions were made for the smaller network sizes of around 5 to 20, below 5 the average vote competence will not have gone up enough to achieve as good performance so this makes sense. The number of incorrect decisions then increased for the larger network sizes (**H5**). This is because the benefit of the increasing competence was outweighed by the issue of high powered Gurus (**A9**). The incorrect decisions fell going from 75 to 100 due to the highest power falling while having more competent options.

The number of Gurus fell as the network size increased. This is because a lot of agents would want to delegate but could not as there were no acceptable options in

their small network. It eventually levels out at 44 as this is how many on average would choose to be Gurus even with acceptable delegation options.

Low competence communities were found to suffer more from the issue of average highest power getting much higher numbers. This is because the agents on average are lower competence so would be more likely to delegate, so you end up with much more powerful super voters who can swing votes in the wrong direction. This causes LD to perform worse than direct voting for high network sizes (**H5**, **A9**).

These results highlight that liquid democracy can perform better than direct voting, and does indeed increase the average vote competence, but this better performance is not guaranteed. When lots of voters are aware of each other then Gurus with high powers can start to outweigh the benefit of increased average vote competence. So it is important to configure a liquid democracy well in real life, else the effect of using it can be detrimental to a community. This is where implementing something like multiple delegations as suggested by Gözl et al. (2018) could possibly help, splitting delegations evenly to reduce the issue of super voters.

## 6.3 One Shot: Effect of Changing Alpha Value for Network Sizes Experiment

### 6.3.1 Results of Network Size Experiment for 3 More Alpha Values

Experiments were completed to measure the effect of increasing how much higher another voters perceived competence must be for a Delegator to be able to (or be willing to) delegate to them. This was achieved by using 3 different values for  $\alpha$  in equation 4.4; recall that the original value for all previous experiments was 0.02.

The network size experiment with Med competence communities was repeated for all the new alpha values to see the effect. For each of the 4 alpha values (including the original) and for all of the network sizes table 6.7 shows the number of incorrect decisions, table 6.8 shows the average vote competence, table 6.9 shows the average highest power, and table 6.10 shows the average number of Gurus.

The incorrect decisions for each alpha value are shown in figure 6.6. For the first 3 values (0.02, 0.05, and 0.1) performance for liquid democracy with lower network sizes gets better as alpha increases, and as the network sizes get bigger they all start to perform worse and end up with high numbers of incorrect decisions. However, when alpha is increased to 0.2 it takes longer for the performance to increase as network size is increased when compared to 0.1, and at most only performs as well as 0.1, never

One Shot Incorrect Decisions				
Network Size	alpha 0.02	alpha 0.05	alpha 0.1	alpha 0.2
1	692	692	692	692
2	484	479	448	580
5	387	352	250	364
10	400	337	263	284
15	372	349	296	290
20	371	344	298	297
25	414	337	336	334
50	448	435	385	545
75	461	471	457	753
100	445	401	459	625

Table 6.7: One Shot Results: Incorrect Decisions by Network Size for Different alpha values

One Shot Average Competence				
Network Size	alpha 0.02	alpha 0.05	alpha 0.1	alpha 0.2
1	0.579804	0.579804	0.579804	0.579804
2	0.608035	0.604481	0.597728	0.587386
5	0.644596	0.643633	0.634672	0.608558
10	0.664146	0.667787	0.666701	0.636873
15	0.672424	0.678106	0.682489	0.658651
20	0.677207	0.684267	0.691609	0.675606
25	0.680837	0.688705	0.697484	0.688382
50	0.68759	0.697684	0.711331	0.723299
75	0.68981	0.701737	0.717457	0.736874
100	0.691261	0.702753	0.719942	0.743419

Table 6.8: One Shot Results: Average Competence by Network Size for Different alpha values

One Shot Average Highest Power				
Network Size	alpha 0.02	alpha 0.05	alpha 0.1	alpha 0.2
1	1	1	1	1
2	3.3217	3.0108	2.6896	2.1826
5	5.5962	5.1464	4.6396	3.9583
10	8.2146	7.9634	7.624	6.82
15	9.6987	9.6838	9.7265	9.2987
20	10.78	11.0142	11.351	11.682
25	11.6065	12.028	12.6909	13.7274
50	14.3633	15.4445	17.0872	22.2105
75	15.3603	17.2576	19.9117	27.969
100	14.8205	16.3771	19.4202	28.1511

Table 6.9: One Shot Results: Average Highest Power by Network Size for Different alpha values

One Shot Average #Gurus				
Network Size	alpha 0.02	alpha 0.05	alpha 0.1	alpha 0.2
1	100	100	100	100
2	72.8493	81.2685	89.9373	97.1578
5	52.22	58.1527	70.863	89.5307
10	46.9741	49.3135	56.9168	79.5515
15	45.6169	46.9402	51.3781	72.0675
20	45.0599	45.9016	48.8543	66.4821
25	44.7051	45.3025	47.452	62.3014
50	44.0101	44.3317	45.2023	51.9683
75	44.011	44.0808	44.5736	48.4324
100	43.9008	44.0335	44.3174	46.9885

Table 6.10: One Shot Results: Average #Gurus by Network Size for Different alpha values

significantly better. For higher network sizes communities with an alpha of 0.2 perform much worse than all the lower tested alpha values.

As visible in figure 6.6 communities with all 4 alpha values make fewer incorrect decisions with liquid democracy at network size 2 when compared to direct. In fact the first 3 alpha values all do better with liquid democracy at any network size (2+) compared to direct voting (1). With alpha at 0.2 however liquid democracy does worse than direct voting with higher network sizes (at least with network sizes 75 and 100).

The best performance is achieved by alpha 0.1, at a network size of 5, with only 250 incorrect decisions (down 442 from direct voting, 121 fewer than the best achieved with the original alpha of 0.02). The worst performance is achieved by alpha 0.2, at a network size of 75, with 753 incorrect decisions (61 more than direct voting).

A graph of the average vote competences across network sizes for all alpha values is given in figure 6.7. The graph shows the average vote competence increases as the network size increases for all the alpha values, with the higher the alpha value the higher the average vote competence at network size 100. Note however, that the average competence starts to increase at a slower rate for alpha 0.2 compared to the other values.

The average highest power across the network sizes is shown in figure 6.8. The lines for each alpha value are similar, but for network sizes of 20 and above the higher the alpha value the higher the average highest power, with a big increase for alpha 0.2.

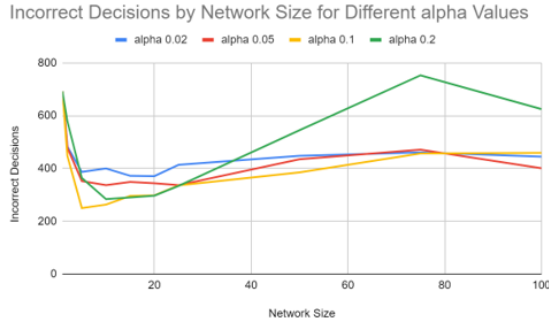


Figure 6.6: Graph of Incorrect Decisions by Network Size (alphas)

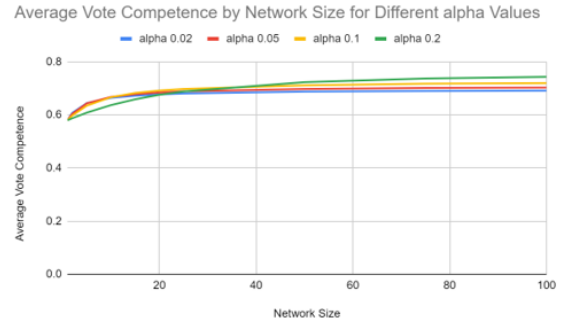


Figure 6.7: Graph of Vote Competence by Network Size (alphas)

For network sizes below 20 the rate of increase for highest average power is slower the higher the alpha value is.

The average number of Gurus across the network sizes is shown in figure 6.9. For 0.02, 0.05, and 0.1 the rate of decrease is lower the higher the alpha value is, but they all level out to the same values of around 44. The rate of decrease for alpha 0.2 is much slower and it does not get at low as the others with the average being just under 47 at network size 100.

### 6.3.2 Discussion

Assuming agents will only delegate to other agents who are perceived at least a certain amount higher is a strong assumption, but can be thought of as replicating things such as advertising high competence Gurus so they become more popular. The results of these experiments show highlighting high competence voters is beneficial, as configuring alpha as 0.05 and 0.1 (**A4**) works better than the original 0.02 (**H1**).

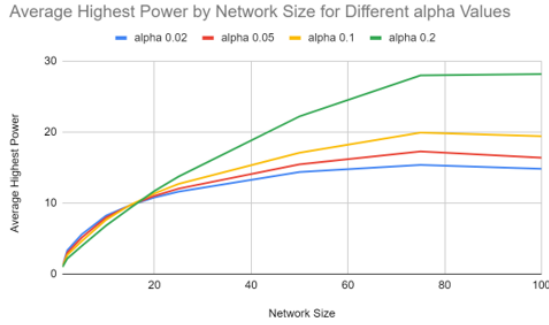


Figure 6.8: Graph of Highest Power by Network Size (alphas)

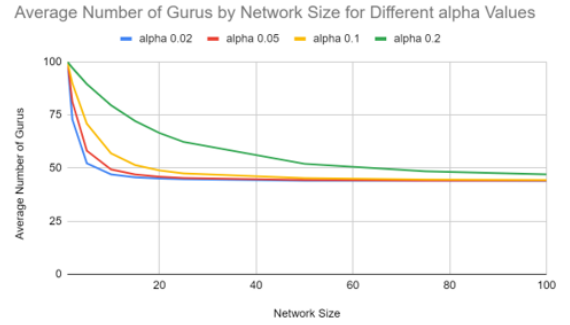


Figure 6.9: Graph of Average #Gurus by Network Size (alphas)

The larger alpha values result in higher average vote competences for higher network sizes due to only allowing high competence difference delegations, but have slightly lower average vote competences at the lower network sizes which is due to some agents not having any acceptable delegations due to the restriction.

Perhaps surprisingly the best improvement in performance for 0.05 and 0.1 over the performance of 0.02 comes at these lower network sizes. The difference at these lower network sizes is the restriction stops agents making delegations to agents who are only slightly more competent and then likely to be wrong with more than just their own vote. Removing the possibility of these bad delegations reduces the number of incorrect votes at low network sizes considerably.

Alpha 0.05 and 0.1 perform about the same as 0.02 for the higher network sizes. It looks like this is because of the average highest power being higher for the higher

alphas at larger network sizes, which is because super voters are more likely as there are fewer acceptable options for Delegators. Even though restricting delegations by increasing alpha can make the average competence higher, it does not mean better results as the benefit can be outweighed by the issue of super voter at higher network sizes.

Eventually increasing alpha loses its benefits as shown by the results for alpha 0.2. For alpha 0.2 the performance never gets much better than it was for 0.1. The number of incorrect decisions takes longer to decrease, which coincides with the average vote competence taking longer to increase, which is due to 20% higher (0.2) being such a big restriction many voters cannot delegate until the larger network sizes.

The highest average power is much bigger for alpha 0.2 at the larger network sizes, which results in LD with alpha 0.2 having super voters and performing worse than direct voting (**A9**).

This big restriction of alpha 0.2 also stops voters who would want to be Delegators from doing so, 0.05 and 0.1 also do this a bit with the average number of Gurus taking longer to fall than for 0.02, but at 0.2 it never gets as low as more voters will never have acceptable delegations. So of course this should never be a real world restriction, but should show the importance of advertising good delegation decisions so there are fewer low competence voters with high weights.



## 6.4 Learning Agents: Direct Voting vs Liquid Democracy

### 6.4.1 Results of Direct Voting vs Liquid Democracy

Next, experiments using single sets of 100 learning agents were performed. These 10,000 round experiments had to be repeated 10 times to get accurate average values. Direct Voting was simulated using a network size of 1. Liquid Democracy was simulated using a starting network size of 10, but as learning agents were used the network sizes of individual agents could grow.

The results of the 10 runs for direct voting are given in table 6.11, with the average statistics at the bottom. The results of the 10 runs for liquid democracy are given in table 6.12, again with the averages at the bottom.

As shown in the tables the average number of incorrect decisions for direct voting was 657.4, slightly lower than but close to the direct voting number in the one shot experiments (table 6.5). The average number of incorrect decisions for liquid democracy was 626.8, only performing slightly better than direct voting, and worse than all the liquid democracy results for all the network sizes in the one shot experiments.

The graph in figure 6.10 models the average vote competence of the 10 liquid democracy simulations across the 10,000 voting rounds. This was created using the CSV output (example in figure 4.22) using data from every individual voting round (100,000

Direct Voting					
Simulation	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
1	9208	792	100	1	0.574694
2	9256	744	100	1	0.575925
3	9438	562	100	1	0.584148
4	9349	651	100	1	0.579815
5	9379	621	100	1	0.580023
6	8841	1159	100	1	0.565526
7	9541	459	100	1	0.587505
8	9273	727	100	1	0.57553
9	9511	489	100	1	0.586519
10	9630	370	100	1	0.59255
Average	9342.6	657.4	100	1	0.5802235

Table 6.11: Learning Agents Results: 10 Runs of Direct Voting Statistics

Liquid Democracy Voting (Alpha 0.02)					
Simulation	Correct Decisions	Incorrect Decisions	Avg #Gurus	Avg Highest Power	Avg Vote Comp
1	9551	449	22.6628	27.112	0.77158
2	9293	707	19.0298	31.0299	0.768882
3	9462	538	19.4145	46.9429	0.827021372
4	9514	486	20.0052	45.4209	0.849689
5	8976	1024	23.4147	29.9011	0.742882
6	9587	413	18.92	30.8168	0.824249
7	9374	626	21.234	28.9368	0.78164
8	9385	615	15.2038	69.5407	0.880785
9	9185	815	14.6003	36.5085	0.792332
10	9405	595	16.068	59.1008	0.853864
Average	9373.2	626.8	19.05531	40.53104	0.8092924372

Table 6.12: Learning Agents Results: 10 Runs of Liquid Democracy (Start Network Size 10) Statistics

in total). This is also how the graphs in figures 6.11 and 6.12 were created. The CSV files are of course too big to be displayed in this paper.

The average vote competence across the rounds in figure 6.10 is modeled against a straight line representing the average vote competence from the 10 direct voting simulations. The direct voting vote competence of course remains the same as there are no delegations; its value is 0.58 as modeled by the gamma distribution. The liquid democracy average vote competence quickly rises in the early rounds from a similar value to constantly above 0.75, averaging at 0.81 (23% more competent than in direct voting).

The average highest power for liquid democracy is shown to quickly rise in the early rounds in figure 6.11 (the highest power in direct voting is of course always 1). It averages out at 40.53, which is much higher than it gets in the one shot experiments (15.36 in table 6.5), sometimes rising above 50%.

The average number of Gurus in the liquid democracy simulations drops to below 25 very quickly (figure 6.12), leveling out faster than vote competence and highest power. The average number of Gurus across the 10 LD simulations is 19.05 (the number of Gurus is of course always 100 in direct voting).

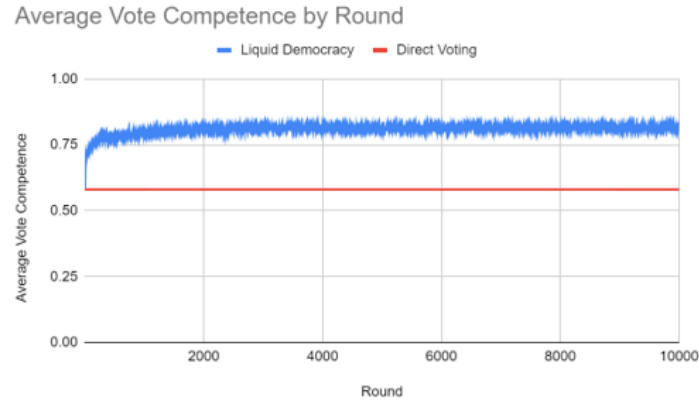


Figure 6.10: Graph of Average Vote Competence from Rounds of the 10 LD runs vs Average Competence of 10 Direct Learning runs

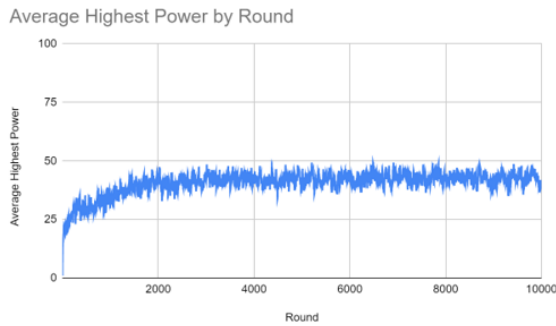


Figure 6.11: Graph of Average Highest Power from Rounds of the 10 LD runs

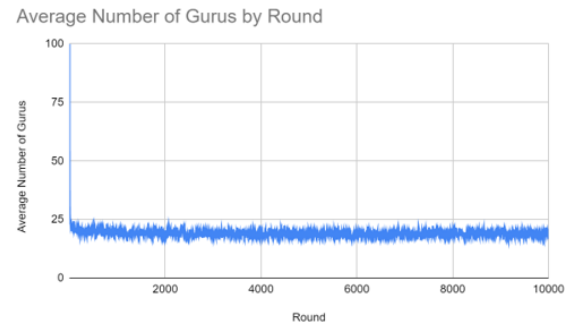


Figure 6.12: Graph of Average #Gurus from Rounds of the 10 LD runs

### 6.4.2 Discussion

The experiments with learning agents showed that single communities using Liquid Democracy that can learn about good delegation options and new delegation options do perform better on average than those using direct voting (**A5**), but only slightly and not as well as the one shot experiments suggest.

The average vote competence of the communities quickly rises to over 80% (**A8**), which shows even though perceived competence is typically not exactly true competence that is must get close quickly as agents start to make good high true competence delegations. Interestingly the average number of Gurus quickly falls to below an average of 20, where it was around 44 in the one shot experiments. This is likely due to having multiple rounds with the same agents where the med competence agents who chose not to delegate initially do eventually try delegating and stick with their Guru while they perform well giving a lower average number of Gurus.

The clear cause of the performance not being amazing can be seen looking at the average highest power, averaging out at 40.53 and rising over 50 on multiple occasions. When it is over 50 you have the worst case of super voter, as they control 50% of the vote so the decision is theirs much like a dictator, but even over 40 is high enough to typically swing the vote either way, so even with a higher competence the performance is barely better than direct voting (**A9**).

While there being fewer Gurus may partially contribute to this large average highest

power, the main cause can likely be put down to the simulation real life learning. The learning means agents can increase their local network size through finding a new Guru through transitive delegations that they did not know before, and as found in the one shot experiments large network of agents who are aware of each other can lead to bad performance. So, lots of agents all find out about each other through different delegations, a few agents are found to be of higher competence and receive more delegations, and the most popular then receives more due to their popularity becoming a super voter. This may not even be down to direct delegations alone to the super voter, but transitive delegations from different local networks could all lead to a single Guru over time.

Of course this could easily be a problem in the real world, as even though many communities are bigger than 100 voters there are things like social media where Gurus can make themselves known to a huge portion of the voters. If super voters are made this way based on popularity you not only have the issue of a super voter, but the Guru may not even be the most competent. So again the importance of pushing good delegations was explored by changing the alpha value.

## 6.5 Learning Agents: Liquid Democracy with Different Alpha Values

### 6.5.1 Results of Learning Agents using Liquid Democracy with Different Alpha Values

As increasing the alpha value saw beneficial results in the one shot experiments, and liquid democracy with alpha 0.02 did not perform much better than direct voting, the learning agents liquid democracy experiments were repeated with 3 higher alpha values. The average results for direct voting and LD with the 4 different alpha values are shown in table 6.13 (they all had individual tables too but these have been omitted to meet space restrictions).

As the alpha value increased the number of incorrect decisions decreased. Even when increasing from 0.1 to 0.2 performance was better on average, albeit by only 5 fewer incorrect decisions, but this was not the case in the one shot experiments. All of the LD runs did better than direct voting on average, but LD with alpha 0.2 did the best with only 367.2 incorrect votes which was 290.2 fewer than direct voting on average.

As the alpha value in LD was increased, the average vote competence decreased each time despite performing better. LD with alpha 0.2 had an average vote competence over 20% less than with alpha 0.02.

Voing	Avg Correct	Avg Incorrect	Avg #Gurus	Avg Highest Power	Avg Vote Comp
Direct	9342.6	657.4	100	1	0.5802235
LD (Alpha 0.02)	9373.2	626.8	19.05531	40.53104	0.8092924372
LD (Alpha 0.05)	9448.2	551.8	25.91305	24.40297	0.7570466
LD (Alpha 0.1)	9627.8	372.2	46.80266	14.67397	0.7128123
LD (Alpha 0.2)	9632.8	367.2	91.27004	5.23866	0.6068798315

Table 6.13: Learning Agents Results: Average Statistics from 10 runs of LD with learning agents with different alpha values

The average highest power also decreased as the alpha value was increased, going down from 40.53 with alpha 0.02 to only 5.24 with alpha 0.2.

The average number of Gurus in the LD runs increased at an exponential rate as the alpha value was increased. For the first 3 alpha values the majority of agents were Delegators, but for 0.2 the majority were Gurus. In fact the average number of Gurus for alpha 0.2 was 91.27 which means there was only 8.73 Delegators on average.

### 6.5.2 Discussion

Liquid Democracy with all of the higher alpha values performed better than the original 0.02 configuration on average, all performing much better on average than direct voting, showing a well configured LD can do significantly better than direct voting (**H2**). In fact, the higher the alpha value the fewer incorrect votes there were, even when going from 0.1 to 0.2, although there was not a significant decrease here.

The average highest power decreased with the increase of alpha, which could be down to fewer agents unnecessarily delegating to agents who are barely more competent than them, or down to fewer Gurus being learned about transitively because of



the restriction. The average number of Gurus also increased, this will be due to fewer agents being able to delegate on average because of the restriction.

Perhaps surprisingly, even though on average the performance increased, the average vote competence actually decreased as the value of alpha went up. This will be down to high competence agents not being able to delegate to higher competence agents because the difference is not high enough, and because low competence agents find fewer new Gurus with higher competences through transitive delegation as the highest competence agent in their network does not delegate. This shows that while increasing the vote competence may sound good it can be more important to stop super voters even if they are highly competent.

Interestingly even though LD with 0.2 does the best on average, it is barely liquid democracy anymore. The whole point of LD is anyone can delegate to anyone, but with alpha 0.2 there are over 91 Gurus on average, barely any voters delegate, but not because they do not want to, it is because they cannot. Even with 0.05 and 0.1 the average number of Gurus rises, but 0.02 shows that these agents have been forced to do this by restriction rather than by choice. So while the contribution of this experiment is that it does show LD can be better than direct voting, restricting is clearly not the way to do it.

So, the important thing is to avoid super voters, but also provide every agent with good delegation options even if they do not know them personally, but then also stop these options from becoming super voters. In bigger communities advertisements for

Gurus could be used so if a voter does not know any good delegation options they can use one they find online or on TV. When it comes to stopping super voters, both in local networks and in the global community with the advertised voters new mechanisms clearly do need to be introduced, such as advertising to voters to become Gurus themselves and care about their power in the vote (Zhang and Grossi, 2021), or allowing multiple potential delegations and having the mechanism choose the one that best spreads the vote (Gölz et al., 2018).

These results from all of the experiments show liquid democracy is worth investing time in to set up the right way, but more mechanisms need to be put in place to ensure good performance.

# Chapter 7

## Conclusion

### 7.1 Contribution

This project has contributed towards the idea of Liquid Democracy through producing simulation software that visualises how a community would vote using such as system. The visualised simulations should make the concepts of liquid democracy such as Gurus, Delegators, delegations, and power clearer to users of the software, giving detail down to the individual voter level.

The created software has provided a lot of data and has the ability to provide a lot more that creates a lot more discussion about successfully using liquid democracy. The collected data shows that liquid democracy can perform better than other voting systems, but also shows there can be issues. The data and discussion could be used to help find the right mechanisms and configurations to use to implement liquid democ-

racy into a real community and get good results from it, getting as much as possible from the benefits while avoiding the potential problems, such as super voters.

The created Liquid Democracy Simulation software has been made open source on GitHub (appendix A) so it is free to be extended by anyone interested in the idea. This means if it is adopted by others that all the proposed mechanisms and options can all be modeled and experimented with in a single piece of software.

## 7.2 Future Work

The only big limitation of this study was time. There are many more experiments the software is capable of running, and much more functionality that could be added. The experiments with the software in this paper focused mainly on medium competence voters with medium values. There were a few experiments with low competence voters but this could have been taken further and different mixtures of values could have been experimented with. To make the simulations more like real life the agents could have had different competence levels for many different vote topics and voter could have delegated to different Gurus for these different topics.

Liquid Democracy is a growing field and there is a lot of recent work where models have been made that could be incorporated into the simulation software. Agents could have an added value of how much they care about power, such as in the work of Zhang and Grossi (2021), whether it be about how much they have or maybe others having

too much. Agents could be able to choose multiple delegates, like in the model by Gözl et al. (2018), which could be used to even out power. There is a lot of new research into liquid democracy, and a lot of it should be included here to find the best possible model for a good liquid democracy.

It would also be good to consider different voting models. The model used here was binary, only two options, with one wrong and one right. While a good basis to show the potential benefits of liquid democracy, a lot of communities may have more options in their votes and this could be made an option for the simulations. Also in the real world there are many votes where people may disagree with what is the best outcome: some voters may vote for personal gain against the best outcome for the community as a whole. The software could be extended to include all of this.

## 7.3 Overview and Reflection

Liquid Democracy is a proposed voting system that some have said is much more democratic and fair than representative voting, while having the potential to give better results than direct voting. Yet despite having all these noted benefits it has not been heavily adopted, and in some cases where it has been the results have been poor with the communities succumbing to issues that it can cause.

The focus of this project was to provide a clear model of liquid democracy which

displays that it can indeed work, but making clear the issues that exist and how they can arise. The simulation software created has achieved this goal. It can be configured in many ways that shows the freedom of directly voting and delegating, shows it can perform well, but also shows what can go wrong.

However, there are many more potential mechanisms that could potentially improve performance that users will need to understand. The software in its current state is a start in making clear how to correctly implement liquid democracy in the real world, but there is much more to be done.

# Bibliography

- Anshelevich, E., Fitzsimmons, Z., Vaish, R. and Xia, L., 2020. Representative proxy voting. *Corr*, abs/2012.06747.
- Behrens, J., Kistner, A., Nitsche, A. and Swierczek, B., 2014. The principles of liquid-feedback. Berlin.
- Blum, C. and Zuber, C.I., 2015. Liquid democracy: Potentials, problems, and perspectives. *Journal of political philosophy*, 24(2), pp.162–182.
- Brill, M. and Talmon, N., 2018. Pairwise liquid democracy. *Proceedings of the twenty-seventh international joint conference on artificial intelligence*, IJCAI-18, pp.137–143.
- democracy.space, 2021. Democracy space — digital voting platform [Online]. [Online], ([Accessed: 02- Aug- 2021]). Available from: <https://democracy.space/>.
- Ernst, D., 2016. What is liquid democracy? [Online]. [Online], ([Accessed: 02- Aug- 2021]). Available from: <https://medium.com/liquid-democracy/what-is-liquid-democracy-b354801f251b>.
- Escoffier, B., Gilbert, H. and Pass-Lanneau, A., 2020. Iterative delegations in liquid

- democracy with restricted preferences. *Proceedings of the aaai conference on artificial intelligence*, 34(2), pp.1926–1933.
- Green-Armytage, J., 2014. Direct voting and proxy voting. *Constitutional political economy*, 26(2), pp.190–220.
- Gölz, P., Kahng, A., Mackenzie, S. and Procaccia, A.D., 2018. The fluid mechanics of liquid democracy. In: *Christodoulou g., harks t. (eds) web and internet economics. wine 2018. lecture notes in computer science*, 11316, pp.188–202.
- Harding, J., 2021. Proxy selection in transitive proxy voting. *Social choice and welfare*.
- Hardt, S. and Lopes, L.C.R., 2015. Google votes: A liquid democracy experiment on a corporate social network. *Technical disclosure commons, defensive publications series*, 79.
- Kahng, A., Mackenzie, S. and Procaccia, A.D., 2021. Liquid democracy: An algorithmic perspective. *Journal of artificial intelligence research*, 70, pp.1223–1252.
- Kotsialou, G. and Riley, L., 2020. Incentivising participation in liquid democracy with breadth first delegation. *Arxiv*, abs/1811.03710.
- Mendoza, N., 2015. Liquid separation: Three fundamental dimensions within liquid-feedback and other voting technologies. *Jedem - ejournal of edemocracy and open government*, 7(2), pp.45–58.
- Price, D.D.S., 1976. A general theory of bibliometric and other cumulative advantage processes. *Journal of the american society for information science*, 27(5), pp.292–306.



- Simulation, Ernst, D. and Ray, C., 2019. Simulation 428 david ernst & cy ray - liquid democracy [Online]. [Online], ([Accessed: 02- Aug- 2021]). Available from: [https://www.youtube.com/watch?v=sa4waQfI\\_Ug](https://www.youtube.com/watch?v=sa4waQfI_Ug).
- Zhang, Y. and Grossi, D., 2021. Power in liquid democracy. *Proceedings of the aaai conference on artificial intelligence*, 35(6), pp.5822–5830.

# Appendices

# Appendix A

## Open Source GitHub and Advance Settings Screen for Extension

Link: REDACTED FOR ANONYMOUS SUBMISSION

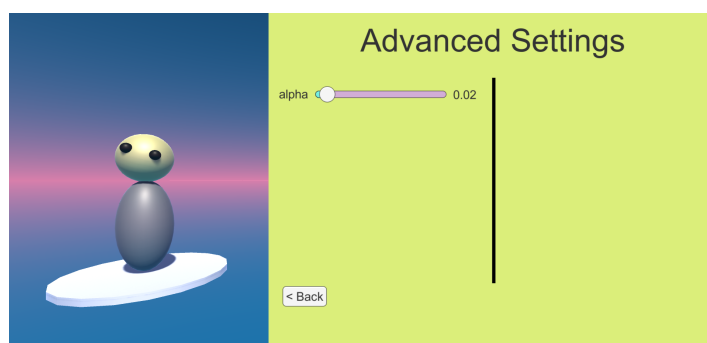


Figure A.1: Advance setting screen that allows alpha to be changed, with free space giving room for more settings, models, and mechanisms in future extensions

## Appendix B

# Video Demo of Liquid Democracy Simulation Program

Link: [Click Here for Video Demo](#)