University of Bath

Literature Review

The Development of a Serious Game to Teach Aristotle's Syllogisms

James Treasure

supervised by
Dr. Willem Heijltjes

April 4, 2017

Contents

1. Literature Review

1.1 Syllogisms

1.1.1 History

Aristotle's work on formal logic was the earliest known study of the topic, which remained at the forefront of academia until the 19th century following Gotlobb Frege's work on first order logic. As syllogisms were so prominent for such a long period of time they have had huge cultural significance on logic in the Western world [Smith, 2016]. Up until the 12th century medieval logicians only had access to a small portion of Aristotle's work with the notable exclusion being Prior Analytics, which contained his work on syllogisms. This period of time is known as logica vetus, or old logic, and it wasn't until the 12th century when Prior Analytics resurfaced in the west that the logica nova, or new logic, began. Immanuel Kant, a prominent 18th century philosopher, went as far as to describe Prior Analytics as "a closed and completed body of doctrine" demonstrating just how highly Aristotle's work on syllogisms was thought of.

1.1.2 What are syllogisms

Aristotle defined his syllogisms as *a discourse in which, certain things having been supposed, something different from the things supposed results of necessity because these things are so.* Put more simply syllogisms are a type of deductive reasoning that, when used on a logical argument, allows a conclusion to be drawn. Aristotle's focus was on categorical syllogisms, essentially a logical argument that contains three categorical propositions. These three categorical propositions in turn are made up of two premises and a conclusion. Each categorical premise is made up of categorical terms, of which each is used twice in the syllogism as a whole. Each categorical premise can be described as a sentence that connects a predicate and a subject by a verb.

$$\begin{gathered} \textit{All } M \textit{ are } P \rightarrow \text{major premise} \\ \textit{All } S \textit{ are } M \rightarrow \text{minor premise} \\ \textit{All } S \textit{ are } P \rightarrow \text{conclusion} \end{gathered}$$

Syllogisms are capable of being applied to any logical argument but to allow them to be compared to each other, they must be represented in a standard form. A categorical syllogism that follows standard form is always structured in the same way, first the major premise, then the minor premise followed by the conclusion.

As created by medieval logicians studying Aristotle's work, the mood and figure of a syllogism provides a notation to represent all the logically unique variations that can occur. The mood refers to the order in which the categorical propositions appear in the syllogism and is represented by four different classes.

- **A** - All A is B → universal affirmative proposition
- **I** - Some A is B → particular affirmative proposition
- **E** - No A is B → universal negative proposition
- **O** - Some A is not B → particular negative proposition

Syllogisms also have a figure that describes the placement of the two middle terms.

	Figure 1	Figure 2	Figure 3	Figure 4
Major	M-P	P-M	M-P	P-M
Minor	S-M	S-M	M-S	M-S

By combining all the different possibilities of mood and figure there are 256 logically unique syllogisms, eg., AOO-2.

All P is M
Some S is not M
Some S is not P

Only 24 syllogisms out of the possible 256 are logically valid, and of these the existential fallacy is broken by 15 of them.

Existential Fallacy

The existential fallacy occurs whenever an argument is invalid because the premises lack existential import. Existential import is an attribute of a categorical proposition that implies the existence of the subject [Hurley, 2005]. There are two schools of thought on the existential fallacy with regards to syllogism, Aristotelian and Boolean.

Traditionally, as laid out by Aristotle, the existence of a subject is assumed when making universal statements. I and O propositions are said to have existential import and in Aristotelian logic I and O propositions must follow from A and E propositions due to subalternation. Subalternation is an inference made between A and I or E and O propositions. It says that if A is true it can be inferred I is true, similarly with E and O. This leads to A and E propositions also having existential import as a proposition with existential import cannot be derived from a proposition without existential import. By considering the syllogism in figure 1.1 we can say that it does not commit the existential fallacy as dogs do exist whereas in figure 1.2 the existential fallacy is committed as dragons do not exist.

However, as modern day logicians and mathematicians often reason about empty sets or imaginary objects the Aristotelian approach to the existential fallacy was problematic. George Boole, a 19th century philosopher, built upon Aristotle's work to create his own Boolean syllogisms to address this issue by stating that universal propositions do not have existential import, thus allowing empty sets [Hill, 2016]. In Boolean syllogisms both figure 1.1 and 1.2 commit the existential fallacy.

All dogs are mammals
All mammals are animals
Some dogs are animals

Figure 1.1

All gorillas are lizards
All dragons are lizards
Some dragons are not gorillas

Figure 1.2

The general rule to reason if a syllogism the commits existential fallacy is that if both premises are universal then the conclusion cannot be particular. In Boolean logic if this rule is broken, then it is a fallacy but in Aristotelian logic the rule must be broken and the critical term must also not exist.

1.1.3 Current teaching

Syllogisms currently feature almost exclusively on philosophy courses around the world, typically as part of units on formal logic. However, Venn Diagrams are taught to children in primary school and set theory features on most maths and computer science degrees. So whilst they are not directly taught about syllogisms, they are already familiar with some of the most common ways to represent them. There are a few different ways that are currently used to represent syllogisms when teaching, with most courses using a mixture of sentential, set theory notation and diagrams. It is common to initially

introduce syllogisms in sentential form, using real world examples as opposed to letters. The advantage of this is it becomes far easier for the learner to visualise the propositions. For example, visualising the statement "all dogs are animals" is simpler than "all A are B" despite being representing the same idea.

However representing syllogisms this way is not the most effective as they can be logically quite complex. As [Larkin and Simon, 1987] explains, sentential form requires inferences to be made, and then those inferences to be held in memory. Working through the syllogism, for example to decide if it is valid, means that these inferences must be continually remembered. As shown by Johnson-Laird [1980], sentential forms of syllogism are more difficult to understand and may be mentally translated into diagrammatic representations internally anyway.

Venn Diagrams

Venn diagrams are the most commonly used diagrammatic representation of syllogisms. The advantage over other diagrammatic forms is that most people have been taught them at a young age and are therefore very familiar with them.

In this representation each circle is a term, which is then filled in to represent the empty set or marked with a X to show at least one member. To achieve the final diagram the two previous diagrams are combined which can then be used to check the validity of the syllogism. It can be seen from table 1.3 that even complex syllogisms are able to be represented with relatively little complexity using Venn diagrams which contributes to their popularity.

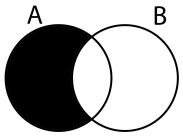
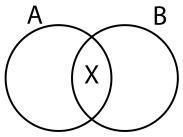
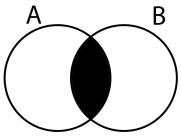
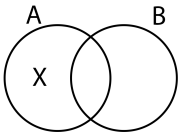
A	I	E	O
			

Table 1.1: Premises represented by Venn Diagrams

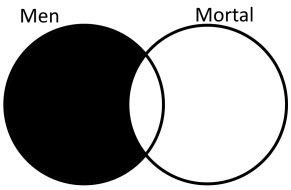
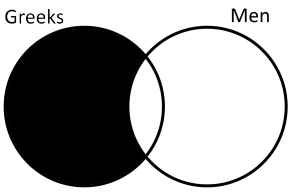
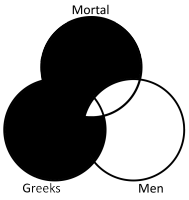
All Men Are Mortal	All Greeks Are Men	All Greeks Are Mortal
		

Table 1.2: AAA-1 Syllogism

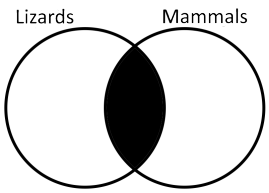
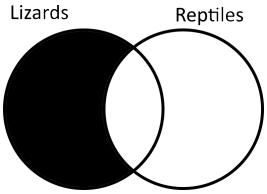
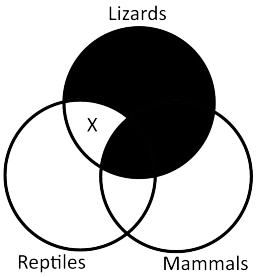
No lizards are mammals	All lizards are reptiles	Some reptiles are not mammals
		

Table 1.3: EAO-3 Syllogism

Euler Circles

Euler circles are another popular way in which syllogisms can be represented. These are a very natural way of representing sets, with Erickson [1978] putting forward the theory that when people are thinking about syllogisms that they are in fact being mentally thought of as Euler circles. As table 1.4 shows, when used to represent the individual premises Euler circles can be very intuitive.

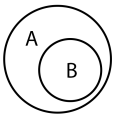
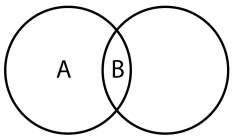
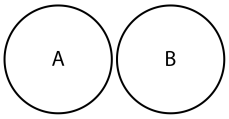
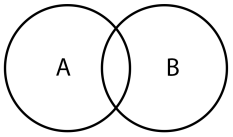
A	I	E	O
			

Table 1.4: Premises represented by Euler Circles

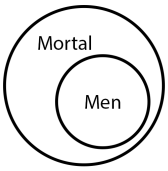
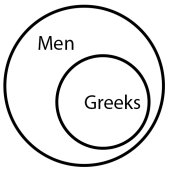
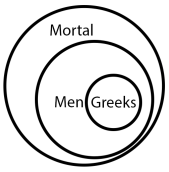
All Men Are Mortal	All Greeks Are Men	All Greeks Are Mortal
		

Table 1.5: Premises represented by Euler Circles

Euler circles also concisely represents simpler syllogisms as table 1.5 shows. However, the problem with this representation arises when more complex syllogisms are constructed. When there are multiple ways to represent the conclusion the result needs more than one diagram to depict all of the possibilities as seen in table 1.6 . It is at this point that the intuitive benefits are quickly outweighed by the added overhead of managing multiple conclusive diagrams.

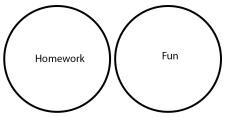
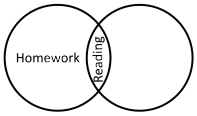
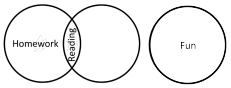
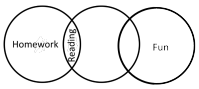
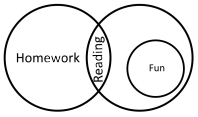
No homework is fun	Some reading is homework	Some reading is not fun
		
		
		

Table 1.6: EIO Syllogism

Linear Diagrams

Linear diagrams are an alternative system that was devised by Englebretsen [1991]. As the name suggests they are linear instead planar like Venn and Euler diagrams. The reasoning behind this was that Englebretsen said it allowed linear diagrams to represent 4 terms, unlike their planar counterparts. Elements are represented by points which are then connected by lines to show sets.

A	I	E	O
			 

Table 1.7: Premises represented by Linear Diagrams

However, as shown by Lemon et al. [1998], linear diagrams do not successfully manage to circumvent the same limitations that planar figures are susceptible to. Lemon et al. [1998] go as far to say that linear diagrams should only be used for trivial objects and relations and that they are unsuitable for more complex sets.

Categorical Pattern Diagrams

As introduced by Cheng [2012], Categorical Pattern Diagrams offer a relatively new approach with regards to representing syllogisms. The aim behind them is to simplify the process of checking validity, better represent syllogisms with more than 2 premises and generally make the diagrams easier to infer from.

The issue with this form of representation is the notation used is not intuitive. Categorical Pattern Diagrams require the person to learn a whole new notation before they can begin to reason about the syllogism that it is representing.

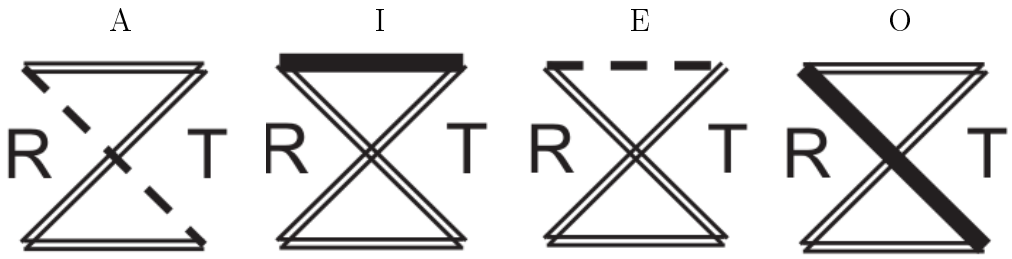


Table 1.8: Premises represented by Categorical Pattern Diagrams

1.1.4 Additional Logic Systems

So far the only formal logic system looked at has been syllogisms, but many more exist. There are number of systems that would be ideal to include in the syllogism game to allow the player to gain a broader understanding of formal logic.

Polysyllogisms

Polysyllogisms, unlike standard syllogisms, can contain more than 3 propositions. This poses a problem for many graphical representations, with traditional circular Venn diagrams being limited to 3 sets. As figure 1.3 and 1.3 show it is possible to represent more than 3 sets using different shapes but as the diagrams become exponentially more complex as sets are added it is quickly becomes impractical to use them.

As reasoned by Cheng [2014], linear and Euler diagrams also struggle with this problem. As both use spatial-overlap it means the diagrams can only represent a series of universal affirmative propositions as further propositions are nested in. As can be seen in table 1.7 and 1.6 there are already multiple ways to represent syllogisms with 3 terms so it is easy to see how it would quickly become unmanageable as the number of propositions increases.

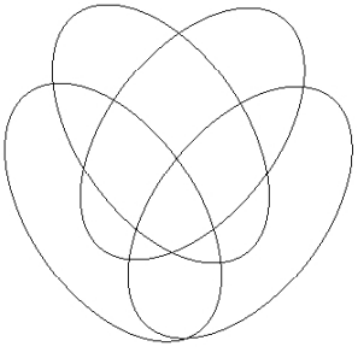


Figure 1.3: 4 Set Venn Diagram

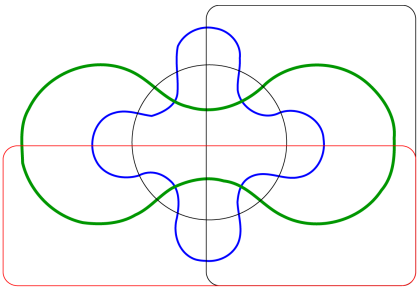


Figure 1.4: 5 Set Venn Diagram

Modal Logic

Modal logic is an extension of classical proposition logic that uses the modal operators *it is possible that* and *it is necessary that* to describe a way in which the rest of the proposition can be true [Zalta, 1988]. Modal logic allows statements to be expressed that would otherwise not be possible in (propositional) classic logic as it is not possible to convey the prospect of events happening. The following are examples of modal propositions:

It is possible that it will snow tomorrow
It is necessary that it is either snowing here now or it is not snowing here now.

The necessity operator is denoted by $\Box$, whilst the possibility operator is represented by $\Diamond$.

Temporal Logic

Temporal logic is a formal system for specifying and reasoning about propositions with regards to time. It is often used in the formal verification of software systems to interpret concurrent events [Lamport, 1983]. Temporal logic can be split into two subsections, linear and branching. Linear temporal logic represents time as a set of paths with each part being a sequence of moments. From any given moment there is only one possible future moment, a unique successor. Branching linear logic represents time as a tree with the root as the present and the branches being the future.

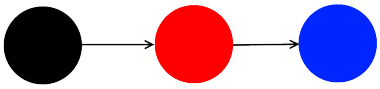


Figure 1.5: Linear Temporal Logic

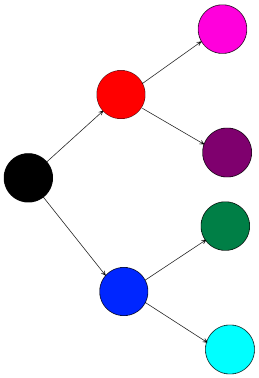


Figure 1.6: Branching Temporal Logic

Set Theory

Set theory is the mathematical theory of well-defined collections known as sets. These sets are made up of objects called elements. Given a set S and an element x it is said $x \in S$ if x is an element of S and conversely $x \notin S$ if x is not an element of S [Bagaria, 2016].

Diagrammatically they appear very similar to syllogisms as both utilise Venn diagrams so it would naturally follow on well to expand syllogisms to encompass set theory. This is especially true when it is considered how syllogisms can be represented using set theory notation as in table 1.9.

Syllogism	Set Notation	Sentential Representation
AaB	$B \subseteq A$	All B is A
AiB	$B \cap A \neq \emptyset$	Some B is A
AeB	$B \cap A = \emptyset$	No B is A
AoB	$B \cap A \nsubseteq B$	Some B is not A

Table 1.9: Set Theory Notation for Syllogisms

1.2 Serious Games

1.2.1 Introduction

Serious Games are a movement within the game industry comprised of software developers and researchers who are using games to educate.

Whilst there is some debate around what exactly construes a serious game, Michael and Chen [2005] define one as any game where entertainment is not the primary purpose, but where instead the focus is on education. They aim to tackle a growing problem whereby educational techniques have not adapted to the needs of the current generation. As Lim [2008] discusses, learning in the classroom has been driven by the national curriculum, with a massive focus on standard and grades. This goes against the belief that to attract the attention of students the focus should in fact be on creating an engaging learning environment. In contrast with previous generations, the younger generation of today have grown up in a world consumed in technology. Oblinger et al. [2005] described them as the *Net Generation*, the generation who always need to be connected, require immediate feedback and crave social interaction. The research of Prensky [2001] explains how this vast amount of technology now experienced in everyday life has led to these newer generations having their minds rewired. These cognitive changes have caused a different set of educational preferences when compared to previous generations, with teaching methods not evolving in order to take this into account.

1.2.2 Why They Are Good For Education

Accessibility

One of the benefits of using games to teach is their accessibility in comparison to other methods. In the United Kingdom access to the internet has doubled over the past 10 years, with 82% of adults now using the internet on a daily basis [Office For National Statistics, 2016]. With such a huge number of people using the internet it means that games hosted on the web have an extremely large potential audience. This broad audience coupled with the widespread appeal of video games allows the crossing of demographic boundaries such as age, gender and educational status [Griffiths, 2002]. Serious Games also have a lower barrier to entry than other educational techniques by allowing a much more "pick up and play" approach to learning with less reliance on prerequisite background knowledge. In the case of syllogisms this would be achieved by using graphical representations to replace the set theory notations. By teaching the concepts first, set theory notation could then be introduced at a more appropriate time in the learning process when the player had a greater understanding of the topic.

Motivation

As discussed by Malone [1981] games can be intrinsically motivating, such that there are no external factors as to why a person is playing other than for their own enjoyment. This is in contrast to typical classroom learning that is usually more extrinsically motivated through grades, exam results and certificates. As Csikszentmihalyi et al. [1997] says, the school attendance would see a sharp drop if extrinsic rewards were removed. Of the two, intrinsic motivation is more resilient than extrinsic motivation and learning that takes place as a result of intrinsic motivation is of higher quality and longer lasting [Kawachi, 2003].

However, it is important to remember that not all games are by definition intrinsically motivating, but need to be structured correctly to achieve this. Malone included challenge, fantasy and curiosity as the three cornerstones of intrinsically motivating serious games. In order to create a challenging game, it should include uncertain goals that are not too easy for the players to complete. Ensuring a game is challenging is crucial as demonstrated by Malone's work surveying children's opinion on 25 different computer games. The results conclusively showed that the most popular games all shared one thing in common; they all contained a defined goal. Similarly in a study carried out by Abuhamdeh and Csikszentmihalyi [2012] on chess players, it was found that the more challenging the game the greater the player's enjoyment. This is important to bear in mind with syllogisms as it is demonstrates the need to add the correct amount of difficulty for the game to be successful.

A fantasy environment in a game is said to "evoke mental images of phyical or social situations not actually present". Malone divides these fantasies into intrinsic and extrinsic, with intrinsic being more interesting and instructional resulting in a greater cognitive effect on learning. Lepper [1988] explains how if learning takes place away from the functionality of the knowledge then this decontextualisation results in the learner becoming demotivated. This can be tackled by using real world examples demonstrating how the concepts are applied. For syllogisms this can just be as simple as using actual examples that exist as opposed to using letters to represent sets. However, as this is not always feasible, Lepper proposed inserting the content of the topic into a fantasy context. Curiosity, like challenge, is stimulated from an optimal level of complexity. Curiosity is also broken down into two categories; sensory and cognitive. Sensory curiosity, as the name suggests, uses lights, sounds and graphics to gain the attention of the participant. Cognitive curiosity is achieved by making the participant doubt their existing knowledge, as people desire to have "good

form" in their cognitive structure. By using this taxonomy when developing a serious game, it is clear that this could be used to engage people who otherwise might not have been interested in learning.

Feedback

Feedback is information regarding a person's understanding of a concept provided with the intention of improving the person's ability. Feedback is a crucial part of the education process and significantly influences the effectiveness of learning.

As stated by Hattie and Timperley [2007], "feedback is one of the most powerful influences on learning and achievement". A further study explored the most effective medium to provide feedback. Computer-assisted, audio and video feedback were found to be most effective forms with the more traditional approach of praise, punishment and providing extrinsic rewards being the worst [Hattie and Timperley, 2007].

One of the main advantages of using games for education is the speed at which feedback can be delivered to the learner which is especially important to the *Net Generation*. The term *fast feedback* was coined by Lumsden and Scott [1988] to demonstrate how this timely feedback removed any uncertainty about progress and allowed any corrective action to be taken during the learning process. However, fast feedback is not a silver bullet and as such the speed at which feedback is delivered should depend on the difficulty of the task. The more complex the task, the greater the benefit of delaying the feedback to allow the information to be processed Clariana et al. [2000].

Feedback during the learning process is known as formative assessments, with summative assessments usually being carried out at the end. As explained by Irons [2007], formative assessments can be beneficial as they enhance the learning process by creating a positive environment which results in greater motivation for learner.

Serious Games lend themselves extremely well to delivering formative feedback to players through in-game features such as progress bars, score count and countdown timers. Feedback can also be provided to the player as they make mistakes allowing the player to engage with the game and keep on track to completing their goals. This can also help the player from becoming frustrated and unable to progress.

1.2.3 Existing Serious Games

Serious Games can be split into two main categories, mini-games and complex games. Mini Games tend to be far shorter, often only taking less than an hour to complete. The focus is aimed at a very specific topic within a subject and is usually browser based. Progression tends to be added through multiple levels, all with similar design but with increasing difficulty. This is the far more common type of Serious Game as they are far cheaper to develop.

In contrast, complex games can take potentially hundreds of hours to complete and cover a very broad topic. The game may contain multiple skills, paths to follow and complex goals.

Serious games do not just apply to education, but are used by a wide variety of different industries. The United States military created America's Army to aid in their recruitment and Microsoft's Flight Simulator is used by pilots in training [Homan and Williams, 1998].

MyMaths

MyMaths is a web based service that is used by schools in over 70 countries, with over 4.5 million users. It is marketed as an online learning platform that provides lessons and homework to school children. It allows them to complete all of this work online and is then automatically marked. As Figure 2.1 shows, MyMaths makes use of fast feedback to provide the player with the results as soon as they have completed the task. However, MyMaths does not provide an immersive or particularly enjoyable experience for players. Lee and Johnston-Wilder [2013] found that while students said they enjoyed using computers to learn, they did not want to use MyMaths. This could be due to that fact that MyMaths is very one dimensional, merely presenting questions in a quiz like format. This does not engage with the player, incite any emotive feelings or provide any fantasy environment. It would more enjoyable if the technologies available were used to make it more interactive.

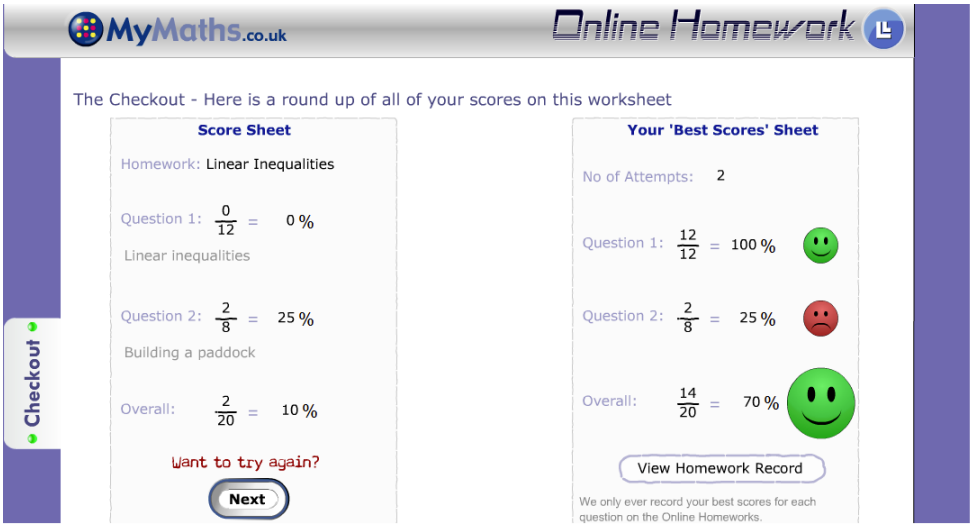


Figure 1.7: A homework exercise in MyMaths

BBC Bitesize

BBC Bitesize is a website maintained by the BBC that provides resources for children in the United Kingdom on a variety of school subjects. Bitesize provides educational information on many subjects, but Maths is one of their main focuses with both static content and a number of educational games. As such Bitesize has a huge number of games available, with 11 KS1 games, 25 KS2 games and 4 KS3 games.

With such a large number of games it allows each game to be very focussed on the topic it is teaching. This is also a disadvantage as for simpler games there is only so much that can be done within a certain topic before the user will start to get bored. As a further criticism the vast majority of games remain at the same difficulty level throughout failing to increase the challenge as the player becomes more proficient. This lack of challenge will lead to players becoming bored and not want to continue playing.

From a technical perspective all of BBC Bitesize games are built using Adobe Flash. This is a serious accessibility problem with major browsers phasing out support for this ageing technology. On top of this Google's Android operating system and Apple's iOS do not support Adobe Flash at all so with most of these games targeted at school children between the ages of 4-14 and with the popularity of tablets and smartphones it possible that they are severely limiting their accessibility.

Figure 1.8: Addition and Subtraction game on BBC Bitesize

From the above examples a trend can be seen that shows that whilst there are a extensive number of educational maths games available on the internet they are aimed almost exclusively at a younger audience. This is perhaps because of the connotations that games are predominantly for children but as discussed previously the benefits of using games for education apply to all ages. This means that most games are based on rather simple, concrete mathematical ideas. There are very few games out there that tackle harder, more abstract concepts such as syllogisms. Also it could be said that the majority of existing games are used to reinforce concepts that have been taught in schools by teachers. The games are aimed at practising those concepts by repeating the ideas over and over. This in turn means that the games themselves are not responsible for actually teaching the core ideas as that is not their purpose. As syllogisms will almost certainly be completely new to anyone playing the game this is something that will have to be focussed

2. Requirements

2.1 Introduction

As stated in section the main aim of this project is to build an educational game to explain formal logic, mainly through the use of Aristotle's syllogisms. The following requirements are separated into functional and nonfunctional requirements. Functional requirements are concerned with how the game should behave in particular situations with relation to the game's functionality. Non-functional requirements place constraints on how the game's functions will complete their tasks, often applying not just to individual functions of the system as a whole

2.2 Sources

2.2.1 Literature Review

The main source of requirements has been drawn from the work in the literature survey. The work carried out that looked into why educational games were so effective allow requirements to be drawn up that ensure the same level of effectiveness when creating this game. Areas such as accessibility, motivation and feedback were extensively researched and the findings have been used to influence these requirements. The discussion of existing serious games and in particular their advantages and disadvantages also provide a source to draw upon.

2.3 Functional

1 **The game must scale dynamically to the screen resolution.**
It is important when supporting devices of different sizes and screen resolutions that the experience remains consistent across them. In the context of the game this relates to the need for the user interface to scale appropriately, font and font sizes being readable across an array of devices and generally no distortion of images.

2 **The game must have multiple levels.**
This allows the game to progress in difficulty, allowing the player to

begin with the basics and work towards learning more advanced aspects syllogisms and logic.

3 **The game must be playable using only the mouse.**
As the game is a simple puzzle game there is no benefit to allowing the user to play with the keyboard in addition to the mouse. By allowing only the mouse to used to play the game it will also make it compatible with devices that do not have a keyboard.

4 **The game should be playable on tablets and mobile phones.**
By allowing the user to interact with the game using only the mouse (finger if on a touchscreen device) and requiring no use of the keyboard it will allow the game to be played across all devices. This aids accessibility as players may not have access to a traditional PC or many want to use a touchscreen device.

5 **The game must feature a tutorial when introducing new concepts.**
A tutorial is crucial to explain the premise and controls of the game. With the focus being on the educational aspect of the game it is important to make interacting with the game and getting to grips with the controls as simple and painless as possible.

6 **The game shall have a leaderboard ranking users.**
A leaderboard showing how the user performed compared to other people who played the game will increase engagements levels. The leaderboard will rank players based on time taken to complete the game as well as acting as a summative assessment informing the user how well they did.

7 **The game must successfully recognise when the player has completed a level.**
The game must be able to accept any valid answer from the user. By not recognising potential end game scenarios it could confuse the user into thinking they are incorrect which would lead to reduced engagement with the game and confusion over what was wrong.

8 **The user shall be able to go backwards and forwards through the tutorials.**
The tutorials will be a crucial aspect of the game, with nearly all the new content and information being delivered through them. It is therefore important that the user can navigate backwards and

forwards through the instructions in case they need to go over a certain piece of information for a second time.

9 **The system shall track time taken to complete each level and moves made**
In order to analyse how effective the game was there will need to be a number of data points recorded. The length of time each level took to complete, along with the number of moves made will be recorded as this allows conclusions to be drawn about how the performance of the user. By combining this data with the initial questionnaire data it will be possible to analyse the effectiveness of the game.

10 **The user shall be able to undo and redo actions.**
An undo and redo function is crucial for a good user experience especially so in systems involving manipulation of graphics as errors are so easily made. As most desktop and many web applications support this functionality, it would detract the user from the experience if this was not an option. By supporting these functions it allows the user to explore the interface with the freedom to experiment without fear of making an irreversible mistake.

11 **The game should provide visual hints to assist the user.**
One of the biggest advantages of using games for education, as mentioned in the literature review, is their ability to provide feedback to the user whilst playing. Therefore it is important to utilise this by providing hints, positive reinforcement and help to the user through on screen graphics during the game.

12 **The user must be able to save the game.**
There should be the functionality for the user to save their place in the game so they can return to it at a later date. This could be useful if the user is interrupted whilst playing the game so they can return to it at a later date and carry on playing from where they left off.

13 **The system shall easily allow new levels to be added.**
With there being 256 syllogisms, although just 24 logically valid, it is important to develop a way to easily add new levels to the game. This is important for both the development of the game as it will allow to easily add new levels and speed up the development phase. This is important as it will allow levels to be added very quickly during the development Also, if the mechanism used to add new levels is simple enough it could be possible for users to create their own levels adding another dimension to the game.

2.4 Non-functional

1 **It must be possible to play the game on all major browsers.**
This requirement is imperative as accessibility is one of the reasons behind building a web based game. It is important to have a consistent experience across all browsers to not detract from the experience of playing the game. The browsers targeted will be current list of browsers as maintained by W3Schools. As most minor browsers are built using the same underlying technology as one of the major ones, provided it works on all of the most popular ones there is a very high chance it will work on the minor ones.

2 **There must be an initial prototype of the game.**
Following an agile methodology when carrying out software development requires the application to be built incrementally. A minimum viable product should be built and then features incrementally added. By using this approach when developing the game it will mean that there will be an initial prototype that can be used to gather user feedback. Supervisor feedback can also be given to steer the project in the right direction and anything that needs to be added can be picked up at the development phase before full user testing takes place.

3 **The game must have an elegant user interface.**
It is important to not crowd the users with an excessive number of buttons and options in the game. As no keyboard controls will be used, the challenge will be to provide a way to operate all controls of the game efficiently.

4 **The game must appear smooth to the user.**
When creating applications that a user interacts with it is important to consider the delays taken between the user interacting with the system to the result being shown on screen. For the game to appear to run without delay, it would ideally react to user input within 100ms. [**?**] explains a delay of less than 100ms will give the user the feeling of the system reacting instantaneously. A delay of longer than 100ms can lead to the user losing their focus on the task and can require feedback from the system informing the user of the delay. User experience guidelines published by Google delve further into this area and say that 16ms is the maximum response time for animations. As this system is a game, that will heavily feature animations, this is important to consider.

5 **The navigation toolbar must remain consistent throughout the game.**
It is important that the navigation bar does not change location throughout the game as it could confuse the user. It is also important that all the options available are present for every level. For example if some levels do not have an undo option it could lead to confusion for the user.

6 **The game must motivate the user to learn about syllogisms**
The game must engage with the user and maintain their attention. It's important to create an environment where the user wants to be playing the game so that the intrinsic desire to play through the levels is strong. It is therefore important to balance the educational aspect with enjoyment and getting the difficulty of the levels correct as to not demotivate the user.

3. Design

3.1 Technologies

Certain technologies must be used when developing applications for the web but there are some areas where there is a choice. The following discusses what technologies will be used in order to create the game and discussion on the potential options available where a choice is available.

3.1.1 HTML

HTML (Hyper Text Markup Language) describes the structure of web pages. Websites created with HTML are then able to be viewed by a web browser. HTML uses tags to tell the web browser how to display certain content with support for links, pictures and video. HTML is only used to define the content of a web page and not how it should be formatted for the end user.

An important part of HTML is the DOM (document object model). When a web page is opened, the browser creates a document model of the page to which objects can be added, modified and removed.

3.1.2 CSS

CSS (Cascading Style Sheets) defines how HTML elements are displayed on a page, essentially allowing formatting to be added such as colour, fonts and layouts. It allows web pages to react to different screen sizes and speeds up the development of projects as styles can be applied to groups of HTML elements.

3.1.3 JavaScript

Initially the web was made up of sites built using HTML and CSS serving static content. As the web progressed there was a desire from developers to create more interactive websites and to respond to these interactions without needing to make relatively expensive calls to a server. This is where JavaScript, a client side programming language, came in. JavaScript allowed elements in the DOM to be interacted with dynamically resulting in a much richer browsing experience.

3.1.4 Web game technologies

There are three potential choices when making a web based game - Adobe Flash, SVG and Canvas. Adobe Flash has been around for the longest, being released in 1996, and became hugely popular in the early 2000s as both a medium to show videos and make games. In 2001 SVG (Scalable vector graphics) was released as alternative followed by Canvas a few years later. The choice of which one to use is not an easy one and can greatly impact the development time, accessibility and appearance of a game.

3.1.5 SVG

SVG (Scalable vector graphics) is a technology that was released in 2001. It uses XML to define vector images that are retained by the browser. SVG is used with the <svg> tag and acts similarly to a HTML tag. It acts on a higher level than Canvas, mentioned later, as objects are stored and remembered in the DOM. This means that all objects created are remembered by the DOM meaning they can be kept track of much easier and natively handle events such as mouse clicks.

3.1.6 Canvas

Canvas is a technology that was originally designed by Apple in 2004. It was later adopted as a web standard and included as part of HTML5, the current HTML standard introduced in 2014. By adding the <canvas> element to HTML it is possible to draw graphics using JavaScript. Canvas allows for the drawing of immediate mode graphics, meaning that once the pixels have been draw to the screen there is no record of these pixels maintained. In turn this means that any time something on the Canvas changes, everything needs to be redrawn.

3.1.7 Canvas Libraries

There are a large number of Canvas libraries out there that aim to simplify many aspects of using Canvas. Fabric.js, Paper.js, Easel.js and Phaser were all looked at with regards to being used in this project. The main features of these libraries include adding support for clicking and dragging objects on the Canvas, simplifying the drawing shapes and assist in animation. The disadvantage of using these libraries is that they are more focussed on traditional games that rely heavily on animations and not the slower puzzle style that is being developed in this situation. They also abstract away the

inner workings of Canvas and can add unnecessary complications to developing what is a reasonably simple game. For this reason it was decided to write the game with no libraries but to just use pure JavaScript.

3.2 Game Design

The overall design of the game is that of a puzzle style with the user working their way through multiple levels of increasing difficulty. The basic mechanics require the user to drag and drop items, usually numbers and shapes, into Venn Diagrams. Each level usually has a statement or fact that must be represented diagrammatically through the correct placement of objects in the venn diagrams.

3.3 User Interface

The goal of the user interface was to be as simple as it possibly could be whilst still allowing all the required functionality to be added. The approach taken with the design was to keep it as clean and minimalist as possible so not distract the player with unnecessary clutter. The relatively small amount of on-screen items also was a benefit when ensuring the game worked well on smaller resolution devices where screen real estate is at a premium. UI mock ups were created that could be shown to users for feedback and acted as a starting point once development began.

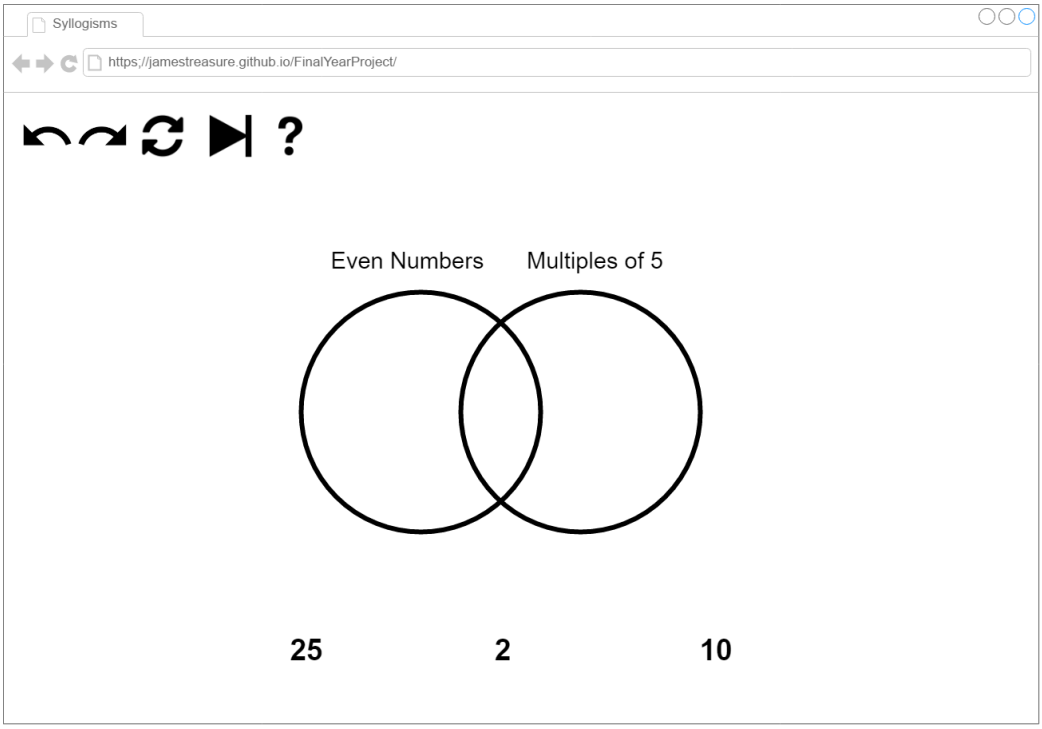


Figure 3.1: UI Mockup

3.4 Control Bar

There are a number of controls that are needed to be present at all times apart from when the user is going through a tutorial. A control bar that remains consistent both in its placement and options available throughout the game will stop the user from feeling confused and allow them to feel confident when interacting with it. The icons chosen were hugely important as there would be no labelling of them so there functionality would have to be shown purely through their image. The icons chosen are based on Google's Material Design icons. It is hoped that through this choice the icons that have been chosen will be familiar to users and as such their function is obvious.

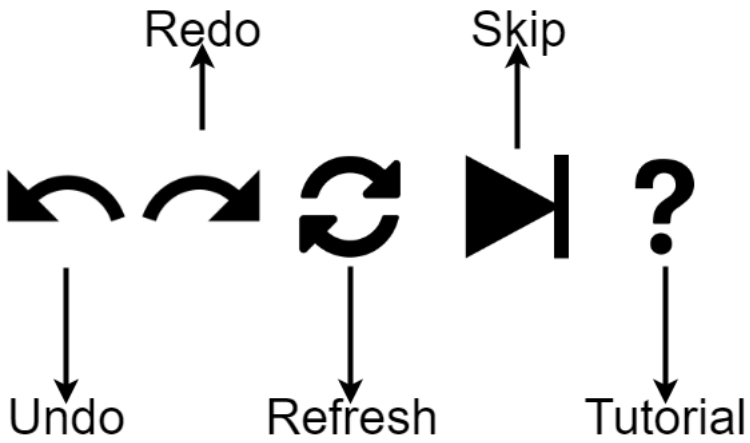


Figure 3.2: Control Bar

3.5 Tutorial Design

There will be a number of tutorials throughout the game so it is important to have a clear and consistent structure to them throughout. This will let the user know when a tutorial is happening so they can pay close attention as it will mean new ideas are about to be introduced. The general structure to the templates involves instructions being displayed on the screen with arrows being drawn to point to the relevant area. There are also forwards and backwards buttons at either side of the screen for the user to navigate their way through the tutorials.

3.6 Level Design

It was very important to design the levels with the correct level of difficulty. If the game started off too hard it would demotivate the player and stop them from wanting to play. Conversely, if the game was too easy and did not introduce harder concepts as the levels progressed their would be no challenge to keep the user interested.

The general idea was to initially have very simple levels that introduced the mechanisms and controls for how the game worked. It also acted as a refreshed on some basic set theory ideas such as empty sets.

From there a level was added that discussed with a simple example how venn diagrams worked. This would most likely have been a refresher as opposed to new content for most users as venn diagrams are a commonly taught way to represent sets in UK schools.

In response to early stage user testing a level was also added that taught the user about empty sets. This was initially an oversight and it was not until users tried to complete the game it became clear that the concept and understanding of empty sets was less prevalent than first thought.

After this there are a number of syllogism levels. The first one is structured with a very tutorial heavy focus. Every step of process from dragging the words into the correct circles to completing the syllogism line by line is aided by the tutorial system guiding the user through the process. After the tutorial the syllogisms get progressively harder with particular syllogisms added, involving the word "some".

As an addition to the game, once the syllogism levels are completed there are a few levels that cover set theory. They aim to teach the basics such as the union and intersection of sets as well as the distributive and associative laws.

4. Implementation

4.1 Undo and Redo

As Canvas does not maintain any record of drawings on the canvas there is no in-built functionality that allows a user to undo or redo an action. In order to achieve this, it must be implemented manually.

There are two ways this can be done; storing everything that is drawn on the canvas in a stack or by capturing the screen as a image and using that image to redraw the canvas. Both options were investigated to see which was the best.

Tests were ran on the execution time of both saving the entire canvas image and using a data structure to store the game state. Saving the entire canvas took 16 milliseconds compared to just 0.1 milliseconds to push the GameState to the stack, 160 times faster. Whilst 16 milliseconds is an acceptable amount of time for an action to take and in isolation does not have an effect on user experience, in combination with other functions that must run on an undo or redo this begins to affect performance. Also as the number of moves increases the number of full canvas images must be stored increases which can lead to memory issues on certain browsers. Another issue with using the image method is that whilst the canvas will revert to its previous image, the underlying state of the game will be incorrect. This is not an issue when using a stack as the location of all the users clicks are stored so the game state is reflected to show this.

The best option was every time a move is made, the current state of all the objects is stored in a data structure and pushed to the stack. When the user presses the undo button, the top object in the stack is popped and everything is redrawn. Despite there being many different objects on the Canvas, the only ones that the user can interact with is the movable text, which can be dragged around, and filling in the circles. Therefore the current state of the game can be represented as:

```
var GameState = function (movableTextArray, clickedInArray) {
   this.movableTextArray = movableTextArray;
   this.clickedInArray = clickedInArray;
};
```

4.2 Clicking on items

As previously mentioned, Canvas uses immediate mode graphics to draw pixels to the screen meaning that once they've been draw Canvas forgets about them. So this means that recognising when a user clicks on an object is not something that is natively supported by Canvas and must be implemented manually. For example, take the level where there are 3 dogs that need to be dragged into the circle. First once the levels loads an array is created which holds the three images as well as properties such as the location of the image and its width and height.

```
[
  {
    "id": "dog1",
    "width": 51,
    "height": 51,
    "y": 857,
    "x": 294,
  },
  {
    "id": "dog2",
    "width": 51,
    "height": 51,
    "y": 857,
    "x": 934,
  }
]
```

Then, any time the user clicks on the screen a check if done to see if that mouse click is inside one of the images in the array. This is done by looping through the array and checking to see if the x and y of the mouse click fall inside the bounding rectangle of the image and then return the index of the image that has been clicked in.

```
function imageClickedOn(x, y, movableTextArray) {
    for (var i = 0; i < movableTextArray.length; i++) {
        if (x >= movableTextArray[i].x &&
            x <= movableTextArray[i].x + movableTextArray[i].
                width &&
            y >= movableTextArray[i].y &&
            y <= movableTextArray[i].y + movableTextArray[i].
                height) {
            return i;
        }
```

```
    }
    return -1;
}
```

4.3 Dragging items

Dragging images and text around the screen is the core of the game, and another feature that Canvas does not provide so again must be implemented manually. First when the user clicks on an image the mouse click event is triggered and all the images are checked to see which one was clicked on, as detailed above. The clicked on image array index is stored as well as a drag offset, which contains the exact location on the image that was actually clicked. When the user then moves their mouse the mouse move event is fired. This checks to see if the drag boolean is true and if it is updates the x and y location of the image to the new mouse location minus the drag offset that was stored. This reason that a drag offset must be stored is that without it the dragging would always take place from the x,y coordinates of the image which in Canvas' case is the top left. So no matter where the user clicked on an imagine they would experience dragging from the top left. Every time the mouse is moved until it is released the x and y get updated and each time that happens everything on the screen needs to be redrawn. There are three ways to do this, setInterval, setTimeout and requestAnimationFrame. Initially, setTimeout was used as it is supported by all browsers. It works by calling a function every specified number of milliseconds. In the table below 1000/60 is used as the interval at which to call the function as this represents 60 frames per second. However, there was often a noticeable amount of delay when lots of intensive tasks were being animated, especially on lower powered devices such as mobile phones.

Therefore, another solution needed to be found. requestAnimationFrame is an API provided by browsers that handle the drawing of animations for both Canvas and DOM-based changes. This allows the browser to perform a number of optimisations such as preventing unnecessary redraws, call pool multiple animations into a single redraw cycle and prevent animations from running on tabs that are not currently open. As table shows, requestAnimationFrame is considerably faster than the other two options. Unfortunately, not all browsers support requestAnimationFrame meaning that a polyfill is needed that allows the browser to fall back to setInterval should requestAnimationFrame not be supported.

```
window.requestAnimFrame = (function(){
  return  window.requestAnimationFrame       ||
          window.webkitRequestAnimationFrame ||
          window.mozRequestAnimationFrame    ||
          function( callback ){
            window.setTimeout(callback, 1000 / 60);
          };
})();
```

Function	Operations per second
setInterval	39
setTimeout	43
requestAnimationFrame	61

Table 4.1: Best way to animate test

4.4 Saving the game

One requirement was that it must be possible for the game to be saved so that the user could return at a later date and carry on playing from where they left off. There are a number of possible solutions for saving the state of a game in the browser.

Previously to achieve this any storage of data would have to have been done through the use of cookies. A cookie is a way to store a small amount of data, up to a maximum of 4kb, in a text file on a user's computer which expire after a certain amount of time.

With the introduction of HTML5 a new method for storing data in the browser was added, localStorage. This new mechanism allows 5MB of data to be stored locally on the browser and is then accessible through any JavaScript that is running on the same domain. It has been implemented here by saving the user's progression after every level that is completed.

So in order to save the game an object is created that holds the current state of the game. This contains both the current level number and the name of the user. Every time a level is completed the object is updated with the new level number and that object is then saved. There is no need to remove the old object as the localStorage API silently overwrites the existing object if the key already exists.

```
\\save object
```

```
localStorage.setItem('gameObject', JSON.stringify(game));
```

The next time the user returns to the web page this saved object can be retrieved.

```
var retrievedObject = JSON.parse(localStorage.getItem('
    gameObject'));
```

The option to resume play is presented to the user through a 'continue' button on the home page. The continue button is initially hidden and by checking that the object retrieved from local storage is not null it can be made visible.

```
if (retrievedObject != null) {
    $("#continueButton").visible();
}
```

4.5 Saving data to database

One of the requirements was for a leaderboard that provides users with the option to enter their name and be able to see how they rank alongside other people who have played the game. This also doubles up as a way to gain an extra layer on analysis being able to take a deeper look at how long levels took and the number of moves that were made.

Initially a REST (REpresentational State Transfer) service was to be created to handle the transport of data. REST uses the HTTP protocol to manipulate data through requests such as GET, which returns data from a datbase and POST which persists data in a database. The REST service would act act as the intermediary between the client side and the database.

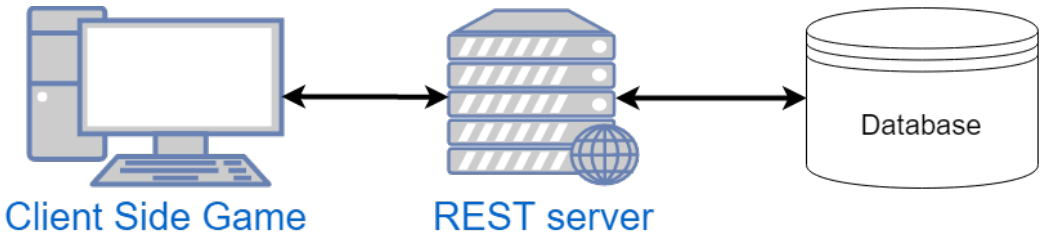


Figure 4.1: Basic REST service architecture

This would require writing REST endpoints to handle such data, using a language such as Node.js, a variant of JavaScript designed to run server side. The problem with this solution is that it requires not only the writing of a REST service but also setting up a database and acquiring a server to host

both of these on. This is overkill for the needs of this project as no sensitive data is being stored, only a few columns of data are needed and speed and number of possible connections that can be sustained are of no concern.

Therefore, for this project Google Sheets is being used as a cloud based database. Google sheets is a web based spreadsheet application, very similar in functionality and appearance to Microsoft Excel. Whilst Google Sheets does not offer the same depth of functionality as desktop based solutions it is perfect for storing a relatively small amount of data with no significant analysis needing to be performed.

Google Sheets does not explicitly support the functionality of using it to act as a database, and as such there are a number of workarounds that must be carried out to make this possible. Google Scripts is a cloud based scripting platform operated by Google that allows JavaScript code to be written that can interact with other Google products, such as Sheets. By using this, it is possible to create what is equivalent to a REST endpoint all hosted within Google's platform. Data can then be posted to this Google Script URL from the game's code that will handle the data and insert it into a Google Sheet.

An object is created that stores all the data that is to be posted, in this case the time and moves taken to complete each level. This object is then serialised for use in a URL query string. This serialised object is then sent to the Google Scripts endpoint via an AJAX request. AJAX (Asynchronous JavaScript and XML) allows client side code to communicate with a web server without needing to reload the page. The asynchronous nature means that the client side code will continue to execute without waiting for a reply, returning only once a reply has been received. This prevents the user from having to wait for the request to be returned.

```
request = $.ajax({
    url: "https://script.google.com/macros/s/
        AKfycbxVGVxoQxNFK7_nxKfglL8yLNUmdPwP2e9j8IM06JY5wzLEdSE
        /exec",
    type: "post",
    data: serializedData
});
```

4.6 Reading from database

Saving data to the database is just one part of the problem, in order to have a functioning leaderboard system it must also be possible to read that data back.

The JavaScript library Sheetrockjs allows Google Sheets to be

interacted with as if they are a traditional database through the use of SQL-like statements. For example, in traditional SQL the query to get the 5 fastest play though times would be:

```
SELECT TOP 5 time_taken, moves_made, levels_skipped
   FROM leaderboard ORDER BY time_taken ASC;
```

In Sheetrock.js a similar syntax is used:

```
$('#spreadsheet').sheetrock({
   url: mySpreadsheet,
   query: "select B,W,X order by X asc",
   fetchSize: 5
});
```

Which get the data from the Google Sheet and fills into the HTML table element with the ID 'spreadsheet'.

4.7 Working across multiple platforms

As this game is designed to run across browsers and mobile devices, it is important that the experience is consistent across all potential screen resolutions. The way that has been chosen to do this is everything drawn on the canvas is relative to the height and width of the screen. For example, when creating the circles to represent the venn diagram the following is done:

```
if (circlesNeeded === 3) {
    //split the screen in to 4 vertical segments
    var segments = canvasHeight / 4;
    //set the radius of each circle
    var radius = segments / 2 * 1.25;

    var overlap = radius / 2;
    circlesArray.push(new Circle((canvasWidth / 2), (segments
        + (segments * 2)) / 2 + (segments / 6), radius));
    circlesArray.push(new Circle((canvasWidth / 2) - overlap,
        (segments * 2 + (segments * 3)) / 2 - (segments / 6),
        radius));
    circlesArray.push(new Circle((canvasWidth / 2) + overlap,
        (segments * 2 + (segments * 3)) / 2 - (segments / 6),
        radius));
```

This allows the user interface to look exactly the same across all screen resolutions. Canvas is supported by the all the major desktop browsers but still has a number of bugs present on Internet Explorer and FireFox. It is

also supported by Android and iOS browsers but some work must be done to actually make it work for touchscreen devices. Touch events must be defined to handle the user touching the screen and dragging their finger along the screen. When using jQuery, the user touching the screen is handled by:

```
$(window).bind('touchstart', function (jQueryEvent)
```

4.8 Floodfill

In order to allow the user to fill in sections of the circles to represent ideas such as the empty set, a flood fill algorithm is needed. A flood fill behaves in the same way as a paint bucket tool in many image editing programs, filling an area of connected pixels with the same colour. There are a number of different algorithms that can achieve this, with recursive and stack based solutions the two most popular. The issue with recursive solutions is that given a complex image there can be a huge number of recursive calls made, potentially causing stack overflow. Therefore, the implementation chosen was to use a stack and a while loop.

- First the x,y location of the mouse click is added to a pixelStack
- While there are elements in pixel stack
  - Pop top value from pixel stack and assign to x and y
  - Travel upwards one pixel at a time until boundary hit or different colour pixel is found
  - Now travel downwards one pixel at a time until boundary hit or different colour pixel is found. This time colouring pixels with new colour as we go.
  - Now check pixel to left to see if it needs colouring
  - If it does need colouring add pixel to stack and set a boolean goLeft to true
  - Now check pixel to right to see if it needs colouring
  - If it does need colouring add pixel to stack and set a boolean goRight to true
  - Move down 1 pixel
  - Look left and right
    - If left or right needs colouring but respective boolean set to true then do not add to stack

    - Else if left or right needs colouring but respective boolean set to false then add to stack
  - Keep moving downwards until boundary or different colour pixel is hit
  - One column is now filled in, but the stack still has pixels in it so go back to start, pop next pixel and run through again.

This algorithm works well but unfortunately has a few problems. Figure 4.5 shows that when using this solution it is not possible to get a *clean* flood fill. This means that the colour black is not filled all the way to the edge and there is a jagged outline due the inherent problem that flood fill algorithms have with anti-aliased edges.

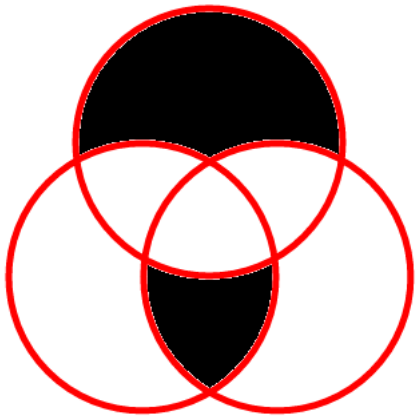


Figure 4.2: Stack based floodfill algorithm

A solution was found by using the library FloodFill2D which implements a more sophisticated floodfill algorithm to specifically deal with the anti-aliasing issue. This library allows a tolerance fade to be set that allows a soft edge to be created at boundaries. However, even that did not completely resolve the problem. In order to get a completely clean appearance to a floodfill more needed to be done. The solution was to create a new Canvas element that sat on top of the one currently being used to draw the circles. This new layer would be used to draw identical circles at a thicket width that would cover up the rough edges on the inner perimeter of the circle. This would not effect the floodfill as the algorithm only floodfills on one Canvas so has no knowledge of these new, wider diameter circles.

4.9 Recognising when level is complete

The location of each text item is found:

```
    var middle = whichCircleIsPremiseIn(level.
       movableTextArray[0]);
    var predicate = whichCircleIsPremiseIn(level.
       movableTextArray[1]);
    var subject = whichCircleIsPremiseIn(level.
       movableTextArray[2]);
```

The location of each intersection is then also stored based on the location of the middle, subject and predicate. The correct answer is for each level is stored in the JSON as so:

```
"correctSyllogism": {
      "subject": true,
      "middleSubjectIntersection": true,
      "subjectPredicateIntersection": true,
      "middlePredicateIntersection": false,
      "middleSubjectPredicateIntersection": false,
      "predicate": false,
      "middle": true
}
```

A new object is created with the same structure as the correct syllogism object. Every time the user clicks in a segment of the circles, which fills it black, the location of the click is stored. This is array of user clicks is used in conjunction with the location of the text to set each location to true or false. From there the correctSyllogism object stored in the game level JSON can be compared with the new object that has been built up by checking the location of the terms to see if they are the same. If both objects are equal then it means that the level has been completed.

4.10 Visual Feedback

As previously mentioned one of the big advantages to using games to educate is that they can provide the user with immediate feedback. There are a number of ways that this have been achieved in this game. When placing items in the circle, the game checks to perform that the placement is valid and if so highlights the circle in green. This immediately shows the user that they have made a correct move. This is achieved by checking each premise is inside the same circle as another premise. If the premise is in its own circle

then a new, green circle is drawn over the top to indicate a valid placement and remains highlighted until the user releases their mouse.

```
function checkPremiseAllInTheirOwnCircle() {
    var middle = whichCircleIsPremiseIn(level.
       movableTextArray[0]);
    var predicate = whichCircleIsPremiseIn(level.
       movableTextArray[1]);
    var subject = whichCircleIsPremiseIn(level.
       movableTextArray[2]);

    if (middle == null || predicate == null || subject ==
       null) {
        return false
    } else {
        return !(middle.equals(predicate) || middle.equals(
           subject) || predicate.equals(subject));

    }
}
```

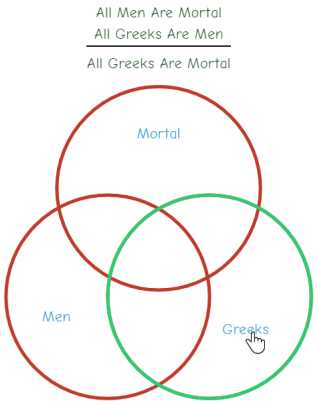


Figure 4.3: Correct Placement

Figure 4.4: Branching Temporal Logic

When completing the syllogism levels there are 3 statements that must be made true; the major premise, the minor premise and the conclusion. When the user makes any of these three true the words font colour of the sentence changes to green.

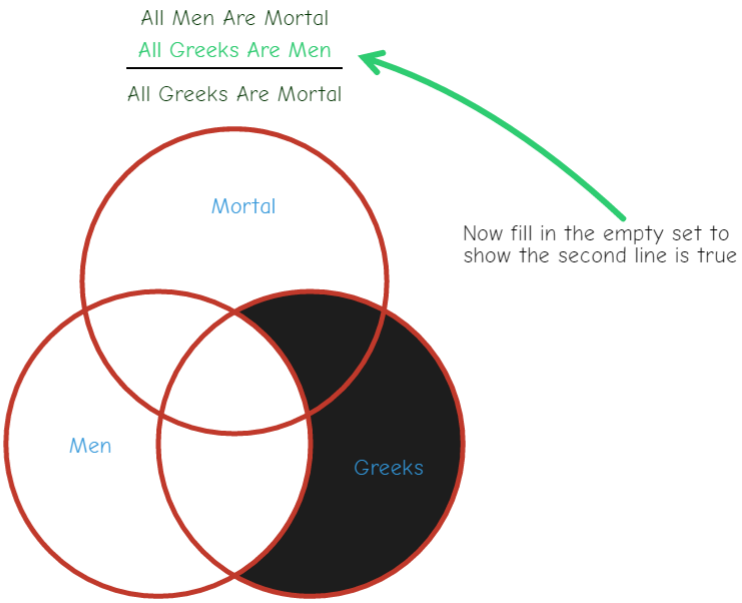


Figure 4.5: Premise Highlighting

4.11 Multiple Canvases

One of the biggest optimisations that can be made to the Canvas environment is to use multiple Canvas elements to act as layers. By doing it is possible to be selective about what items are cleared and redrawn in the animation loop. As mentioned in a previous section, when dragging items across the screen the Canvas is constantly being cleared and redrawn. Initially, when only one Canvas was in use this meant redrawing the circles every time. Drawing 3 circles is obviously negligible but when the ability to flood fill segments of the circle issues began to arise. Initially, when the user floodFilled an area, the X and Y coordinates were saved. Then, when in the animating loop this was cleared every time the coordinates that had been filled were passed to the flood fill algorithm to refill. So on every single run of the animation loop, around 60 per second, the flood fill was being carried out. This was a performance killer and almost instantly ground the game to a halt causing the game to become very laggy.

Tests carried out by show that creating as many as 500 Canvas elements had no noticeable effect on the time it took requestAnimationFrame to complete an animation render cycle. It was therefore decided to separate the game into its individual parts and draw each one on its own Canvas. This resulted in a total of 7 Canvases being used to create the game.

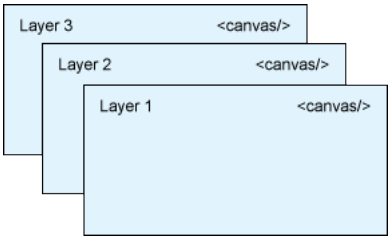


Figure 4.6: Canvas Layering

https://www.ibm.com/developerworks/library/wa-canvashtml5layering/
http://mcgivery.com/how-many-stacked-canvas-elements-is-too-much/

5. Testing

5.1 Test Driven Development

Test Driven Development is a process that involves incrementally developing software by writing unit tests before any functional code is written. The process is as follows:

- Write a unit test
- Run the unit test to ensure that it fails
- Write the bare minimum functional code to make the unit test pass
- Refactor the code

There are a number of benefits to using test driven development when working on large software projects. Müller and Padberg [2003] found that whilst test driven development might result in more overall code being written for a project the total time to actually complete it can be shorter. One of the main benefits is that it provides security when making any major code changes as happened frequently during the development stage of this project. It means that the suite of tests can immediately be re-run and any breaking changes are highlighted which can then be quickly fixed.

As the entire user interface is written exclusively in Canvas and the use of HTML and CSS is very minimal any unit tests would be explicitly testing the Canvas aspect of the code. Writing unit tests to ensure that certain UI elements are drawn, such as circles, text or images is not really possible given Canvas' 'draw and forget' approach. Therefore, all testing of the UI was done manually and through user testing.

There are a number of functions that do lend themselves to be unit tested. Functions that simply take data in and perform checks and operations on it are perfect examples. In the scope of this project this concerns functions that check to see when levels are complete, check to see if user clicks are in a certain place and setting up shapes ready to be drawn. Jasmine, a behaviour driven framework for testing JavaScript code was used to write the unit tests.

```
describe("Array contains another array", function () {
    it("returns true", function () {
```

```
        var array1 = [
            {
                "circleClickedIn": [
                    1
                ],
                "x": 619,
                "y": 420
            }
        ];

        var array2 = [1];

        expect(arrayContainsAnotherArray(array1, array2)).
           toEqual(true);
    });
});
```

Bibliography

Sami Abuhamdeh and Mihaly Csikszentmihalyi. The importance of challenge for the enjoyment of intrinsically motivated, goal-directed activities. *Personality and Social Psychology Bulletin*, 38(3):317–330, 2012.

Joan Bagaria. Set theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2016 edition, 2016.

Peter C-H Cheng. Graphical notations for syllogisms: how alternative representations impact the accessibility of concepts. *Journal of Visual Languages & Computing*, 25(3):170–185, 2014.

Peter CH Cheng. Visualizing syllogisms: category pattern diagrams versus venn diagrams. 2012.

Roy B Clariana, Daren Wagner, and Lucia C Roher Murphy. Applying a connectionist description of feedback timing. *Educational Technology Research and Development*, 48(3):5–22, 2000.

Mihaly Csikszentmihalyi, Kevin Rathunde, and Samuel Whalen. *Talented teenagers: The roots of success and failure*. Cambridge University Press, 1997.

George Englebretsen. Linear diagrams for syllogisms (with relationals). *Notre Dame J. Formal Logic*, 33(1):37–69, 12 1991. doi: 10.1305/ndjfl/1093636009. URL `http://dx.doi.org/10.1305/ndjfl/1093636009`.

J. R. Erickson. Research on syllogistic reasoning. In Russell Revlin and Richard E. Mayer, editors, *Human Reasoning*, pages 39–50. Distributed Solely by Halsted Press, 1978.

Mark Griffiths. The educational benefits of videogames. *Education and Health*, 20(3):47–51, 2002.

John Hattie and Helen Timperley. The power of feedback. *Review of educational research*, 77(1):81–112, 2007.

Hammer Hill. Existential import. http://cstl-cla.semo.edu/hhill/pl120/notes/existential%20import.htm, 2016.

Willem Homan and Kathleen Williams. Pc-based flight simulators: No more games. *Tech Directions*, 58(2):33, 1998. ISSN 10629351. URL `http://search.ebscohost.com.ezproxy1.bath.ac.uk/login.aspx?direct=true&db=bth&AN=1117015&site=ehost-live`.

Patrick Hurley. *A concise introduction to logic*. Cengage Learning, 2005.

Alastair Irons. *Enhancing learning through formative assessment and feedback*. Routledge, 2007.

Philip N Johnson-Laird. Mental models in cognitive science. *Cognitive science*, 4(1):71–115, 1980.

Paul Kawachi. Initiating intrinsic motivation in online education: Review of the current state of the art. *Interactive Learning Environments*, 11(1): 59–81, 2003.

Leslie Lamport. What good is temporal logic? In *IFIP congress*, volume 83, pages 657–668, 1983.

Jill H Larkin and Herbert A Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science*, 11(1):65–100, 1987.

Clare Lee and Sue Johnston-Wilder. Learning mathematicsâĂŤletting the pupils have their say. *Educational Studies in Mathematics*, 83(2):163–180, 2013.

Oliver Lemon, Ian Pratt, et al. On the insufficiency of linear diagrams for syllogisms. *Notre Dame Journal of Formal Logic*, 39(4):573–580, 1998.

Mark R Lepper. Motivational considerations in the study of instruction. *Cognition and instruction*, 5(4):289–309, 1988.

Cher Ping Lim. Spirit of the game: Empowering students as designers in schools? *British Journal of Educational Technology*, 39(6):996–1003, 2008.

Keith Lumsden and Alex Scott. A characteristics approach to the evaluation of economics software packages. *The Journal of Economic Education*, 19 (4):353–362, 1988.

Thomas W Malone. Toward a theory of intrinsically motivating instruction. *Cognitive science*, 5(4):333–369, 1981.

David R Michael and Sandra L Chen. *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade, 2005.

Matthias M Müller and Frank Padberg. About the return on investment of test-driven development. In *Edser-5 5 th international workshop on economic-driven software engineering research*, page 26, 2003.

Diana Oblinger, James L Oblinger, and Joan K Lippincott. *Educating the net generation*. Boulder, Colo.: EDUCAUSE, c2005. 1 v.(various pagings): illustrations., 2005.

Office For National Statistics. Internet access - households and individuals. Technical report, 2016. URL `http://www.ons.gov.uk/peoplepopulationandcommunity/householdcharacteristics/homeinternetandsocialmediausage/bulletins/internetaccesshouseholdsandindividuals/2016`.

Marc Prensky. The games generations: How learners have changed. *Digital game-based learning*, 1, 2001.

Robin Smith. Aristotle's logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2016 edition, 2016.

Edward N Zalta. Basic concepts in modal logic. *Center for the Study of Language and Information Publications*, page 2, 1988.