Topological Tic-Tac-Toe Report

[**Analysis 4**](#_h7dr7vdfwpnt)

[Background 4](#_jgcyxngpsenb)

[Research 5](#_3jv9rjv4shir)

[Search Results 5](#_u9nrbzytyajg)

[Current Programs 8](#_ga2slyq340lu)

[Questionnaire 8](#_nukiai10fo7u)

[Analysis of Responses 9](#_w2ulwnh58n1t)

[Summary 14](#_omfen7qplgew)

[Modelling 16](#_c3kwo8wcykqc)

[Computer Single-Player Algorithm Flowchart 16](#_6obwtrw72g7w)

[Database Entity Descriptions & ER Diagram 17](#_hkmuyd2q0j3g)

[Data Flow Diagram 18](#_1g7cfz3abaxw)

[Data Volumes 18](#_qtmtkqj1tri9)

[Proposed Solution Details 18](#_k9si4dqlmq89)

[Programming Language 18](#_2rezaz9cetfm)

[Option A 18](#_yql8em7m6f58)

[Option B 19](#_gt4bfsq0lhq2)

[Conclusion 19](#_nn7r5katzhxn)

[Graphical User Interface 19](#_j57sdiygffse)

[Option A 19](#_fxmo6c60bzvy)

[Option B 19](#_9j2vdvsvr99j)

[Conclusion 19](#_ooqs637ea777)

[Database 19](#_lsdokcqgt4v1)

[Objectives 19](#_mjffyezb2vgo)

[Core Objectives 19](#_mcjir466bkpu)

[Additional Objectives 21](#_5tvu6dylgbuk)

[**Design 22**](#_70ztscpi5uc4)

[System Summary 22](#_5p9494mv2gdv)

[Menu 22](#_28e71nil721i)

[Settings Screen 22](#_rmnv7qc5o1e4)

[Play Game Screen – Playing a New Game 22](#_742lapov0j0v)

[Load Previous Game Screen 23](#_u7rrdam7bv6k)

[Play Game Screen – Loading Previous Game 23](#_eger3o1o8xol)

[Grid 24](#_akoppzrkjpjd)

[Computer Single-Player Algorithm 24](#_9iuc8ejrl9bb)

[Hierarchy Chart 24](#_5eqdlbqwjce6)

[Classes 24](#_ixytdnjetk7b)

[Button 24](#_g0hned1i5ji5)

[List of Buttons 25](#_7v9fnnxpwfe0)

[Character Input Box 26](#_b17cedl7ng4g)

[Grid 27](#_joy2l92zdjpd)

[Menu 28](#_okzzauqnuxbq)

[Previous Game 29](#_uipwivv66xlq)

[Game 30](#_e1cuehtkoh4o)

[Settings 32](#_ge7wh6ou990t)

[Computer Single-Player Algorithm 33](#_h9e3vo56t6we)

[Class Diagram 34](#_wu3y4kmxed5o)

[Algorithms 36](#_6egjger8csv4)

[Corresponding Square 36](#_6p2ix1ngddmv)

[Adjacent Square 36](#_swh4ligjp90t)

[Get Orientation of Line 36](#_jf208zjtt877)

[Counting Outcome of Each Move 38](#_62btnnhzf50s)

[Get Best Move 38](#_b4n31dacwiar)

[Data Structures 39](#_i2hgkz7uq9z9)

[List 39](#_qzs0tuu3eqlp)

[Dictionary 39](#_8e6d5zo2lq3m)

[Tuple 39](#_ulrwwdv8d3za)

[String 39](#_kb24xocd2d2g)

[Stack 39](#_2aeatyp3jm7n)

[Database Design 40](#_g3mzsv92dros)

[Entity Descriptions 40](#_re6uif93bobg)

[Queries 41](#_klzv6ordy6b5)

[Create Game Relation 41](#_yn2xlqm00qsw)

[Create Move Relation 42](#_6icby3x6barj)

[Insert Record into Game Relation 42](#_bu3fmoxx02sb)

[Retrieve Records from Game Relation 42](#_jsikspfmu20b)

[Retrieve Records from Setttings Relation 43](#_p0lhkehmn5v9)

[Retrieve Number of Wins for a Player from Game Relation 43](#_61ixgcuwyt6h)

[UI Diagrams 43](#_dag6t5r1zae)

[Menu 43](#_vqxge5m4ttaz)

[Settings Screen 44](#_e44zmgcjxpd5)

[Load Previous Game Screen 45](#_j00zpjd3b7l0)

[Game Screen 46](#_ee48ystin9ze)

[**Technical Solution 47**](#_gm1dkezbnbvq)

[Technical Skills 47](#_rcgeqvusyzy7)

[File Structure 47](#_6dh65w79cpa2)

[Project Files 48](#_oo0p1bvqmnxa)

[\_\_init\_\_.py 48](#_2o50ehmodh5k)

[computer.py 48](#_jam3t9mgdfcz)

[game.py 49](#_wl3a5ut4g0jz)

[grid.py 55](#_iz8k6b8w3le4)

[inputs.py 66](#_ce3yl22341t)

[menu.py 69](#_9uhakv3robzk)

[previous\_game.py 71](#_r9ybijfex2t3)

[settings.py 77](#_opcirt63nfok)

[sql.py 83](#_dhm6lp2h77em)

[utils.py 86](#_n9ievhouf8be)

[main.py 87](#_gb94dd686ho7)

[**Testing 91**](#_wvhh9wwnx9q4)

[Test Database 91](#_qn6eh38gocvq)

[Previous Games 92](#_p4t4resezu50)

[Game 1 92](#_fj3c2ta2cff8)

[Settings 92](#_kwnj6agcqhjs)

[Moves 92](#_t1p7b23zugbt)

[Game 2 94](#_4v6xyl9x6oqk)

[Settings 94](#_y19ren11zjpv)

[Moves 94](#_hwa4z0aluz6b)

[Game 3 96](#_fuwq6y5xp10q)

[Settings 96](#_gm10fix3hust)

[Moves 96](#_uat2279c44mn)

[Database 101](#_56xvvxi8ntz)

[Game 101](#_hmtbktdhrlqp)

[Move 101](#_l98grelunvjg)

[Settings 102](#_u1dmdjnk1vwu)

[Queries 102](#_s7n55jp9ho1g)

[Game 102](#_24yrrb25l0i1)

[Move 102](#_c92lqsndunbk)

[Settings 103](#_60pz1mgr5yg7)

[Test Log 103](#_r5ojr5ktlc8u)

[1. General 103](#_z8xkydlaspj3)

[2. Menu 103](#_r91iqbify3d)

[3. Settings Screen 108](#_nkbjzdxu0ktb)

[4. Load Previous Game Screen 118](#_gy9n16a3u1fx)

[5. Game Screen 121](#_ao7k19fvu6fz)

[6. Computer Algorithm 134](#_qqip8p3k4zj1)

[7. Grid 139](#_f7rpkle3qtvm)

[8. Inputs 161](#_d6ukfioecqsx)

[9. SQL 169](#_68gf03012ej)

[10. Utils 172](#_1hfy1khxs9v)

[**Evaluation 174**](#_yxxo2f4d9sk1)

[Objectives Evaluation 174](#_8xp865lkusjr)

[Core Objectives 174](#_4apmhfz3xt1u)

[Additional Objectives 178](#_g5lnmmgzlxw4)

[Feedback Analysis 178](#_mygk0zhpm8fl)

[Overall Solution Effectiveness 179](#_o4fztgmt3u4f)

[Improvement 179](#_6g352up665zq)

[**Appendix 180**](#_4rn3humrfeuk)

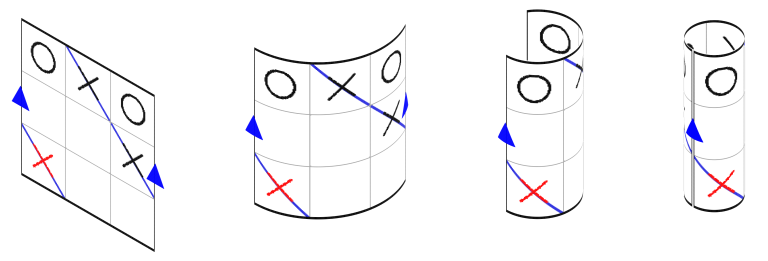
[End-User Feedback 180](#_8deqau1tj917)

# Analysis

## Background

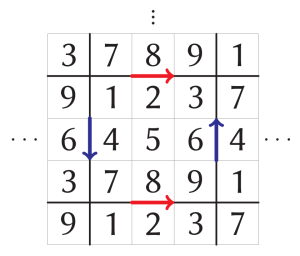
I will create a game (with menu, game and game history screens) that allows users to play tic-tac-toe in different formats. They will be able to vary the topology of the grid, which they are playing on, (e.g., plane, cylinder, mobius strip, torus, klein bottle, projective plane), the size of the grid and length of the line needed to win. Also, they will be able to: play against a computer or another user; undo their moves and possibly load their previous games from a database.

The game will be called Topological Tic-Tac-Toe. This game differs from normal tic-tac-toe inasmuch as the grid can have different topologies. This means that the grid could be folded round such that left and right and/or top and bottom sides are touching. Hence, a winning line of noughts or crosses could exit from one side of the grid and enter onto the other side. This image aids the explanation:



<https://chalkdustmagazine.com/features/topological-tic-tac-toe/> (21/07/2022)

Also, the grid could be twisted before the two sides are joined. For example, the right side could be twisted before being joined to the left side so that the top of the right side touches the bottom of the left side and vice versa. This allows more winning lines to be made than normal tic-tac-toe permits. An adjacency map is set of imaginary squares around the real grid, which display where noughts and crosses would appear when the grid is folded according to its topology. This image aids the explanation. *Notate bene,* parallel lines depict two sides are touching and anti-parallel lines depict that two sides are twisted and touching.



<https://chalkdustmagazine.com/features/topological-tic-tac-toe/> (21/07/2022)

These two articles explain the game in more depth:

<https://my.integralmaths.org/mod/book/view.php?id=78527&chapterid=7269> (21/07/2022)

<https://chalkdustmagazine.com/features/topological-tic-tac-toe/> (21/07/2022)

Topology is a degree-level Maths concept and hence there are not many interactive online tools available to explore it. There are only really guides to physical demonstrations of topology. The aim of this project is to fill that gap.

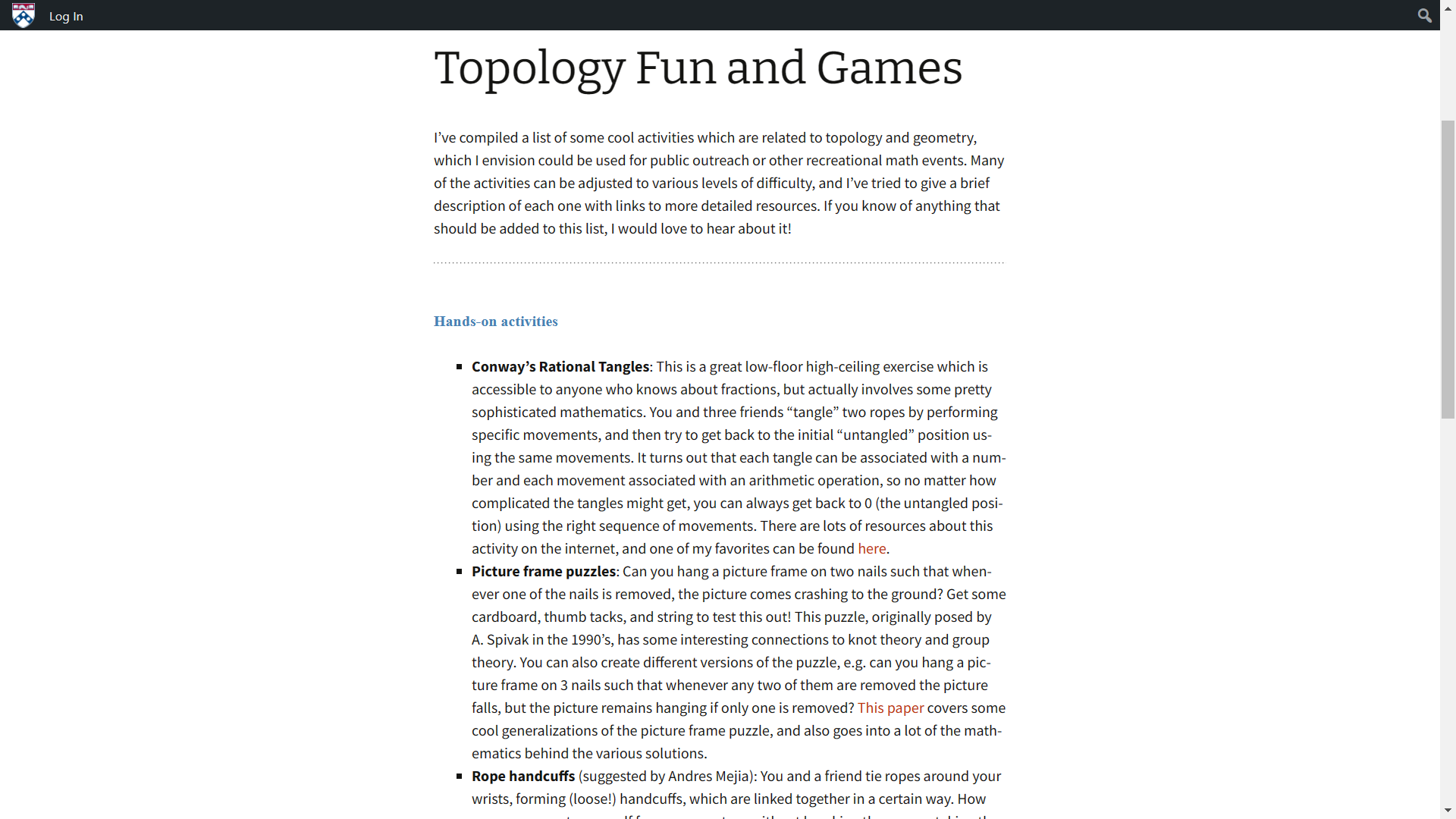
This problem is being solved for Mathematicians or Computer Scientists, who are interested in mathematical topology and would like to play a simple game to test their understanding. I will send questionnaires to my Maths and Computer Science teachers to gauge what they want to see in the game.

## Research

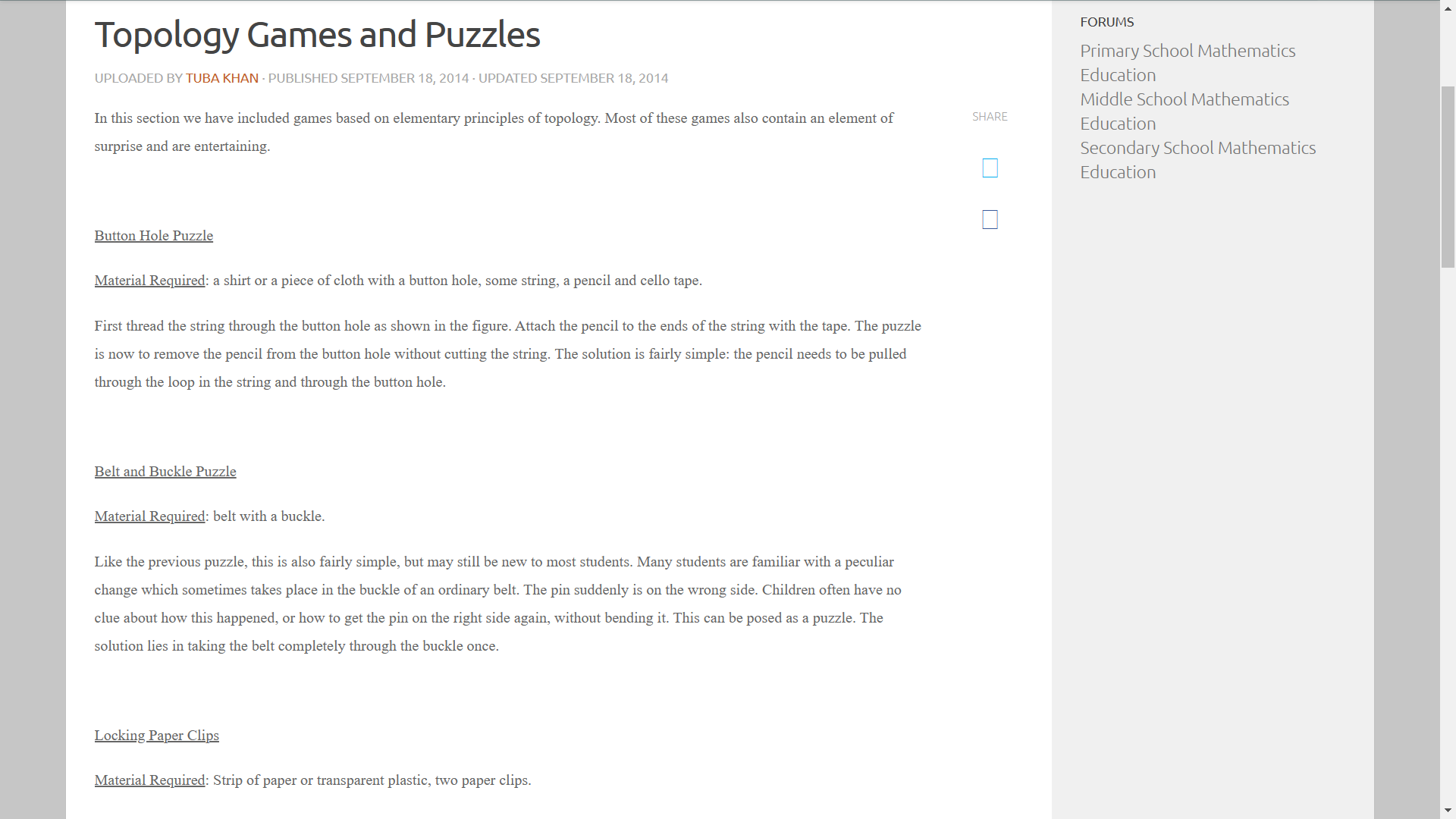
### Search Results

When I searched up on Google “topology maths game”, the top three results were:

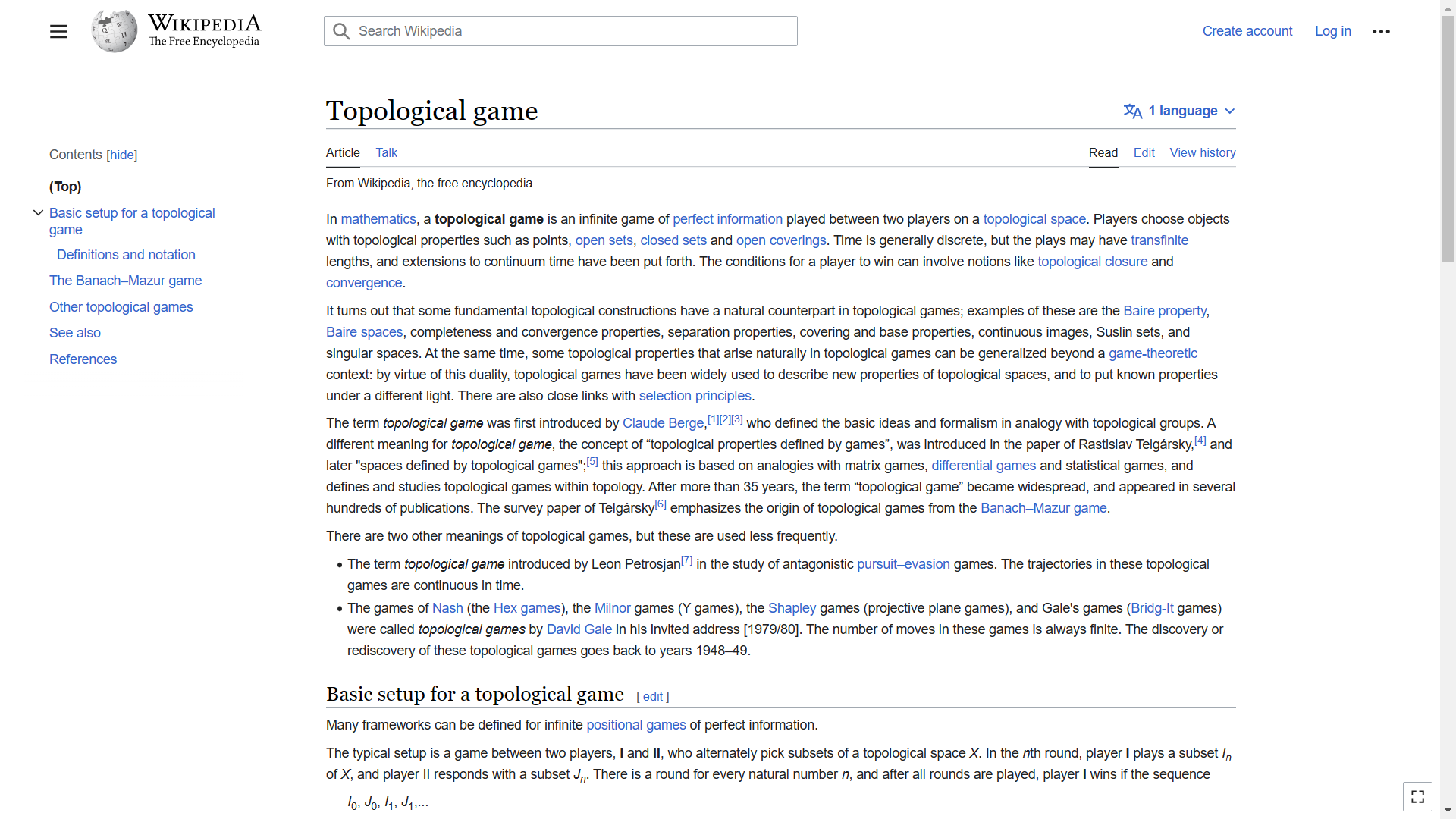
1. <https://web.sas.upenn.edu/callem/topology-fun-and-games/> (20/07/2022)



1. <https://mathedu.hbcse.tifr.res.in/topology-games-and-puzzles/> (20/07/2022)



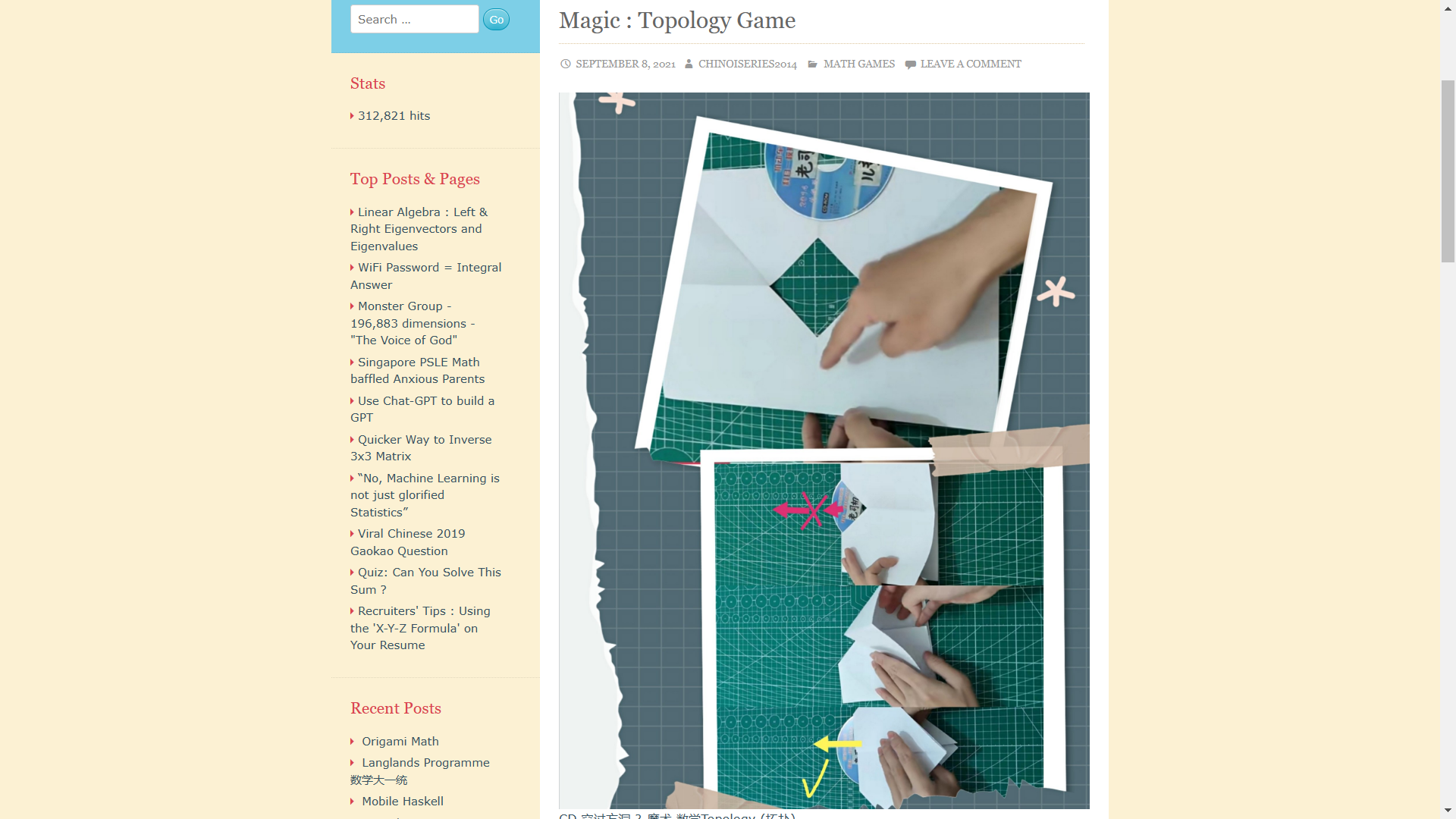
1. <https://en.wikipedia.org/wiki/Topological_game> (20/07/2022)



The top two results contain physical games, which could be played in real-life, and the third result contains information about a topological game theory game.

Searching, up “interactive topology maths game”, the top three results were:

1. <https://web.sas.upenn.edu/callem/topology-fun-and-games/> (20/07/2022)
2. <https://tomcircle.wordpress.com/2021/09/08/magic-topology-game/> (20/07/2022)



1. <https://mathedu.hbcse.tifr.res.in/topology-games-and-puzzles/> (20/07/2022)

Two of the results here are in the top three results above. The second one contains an image with some origami.

I also found this Quora question “Is there a video game to learn topology?”:

<https://www.quora.com/Is-there-a-video-game-to-learn-topology> (21/07/2022)

The top answer suggests that there are many physical games, which have been invented to explore topology, but these are not virtual video games.

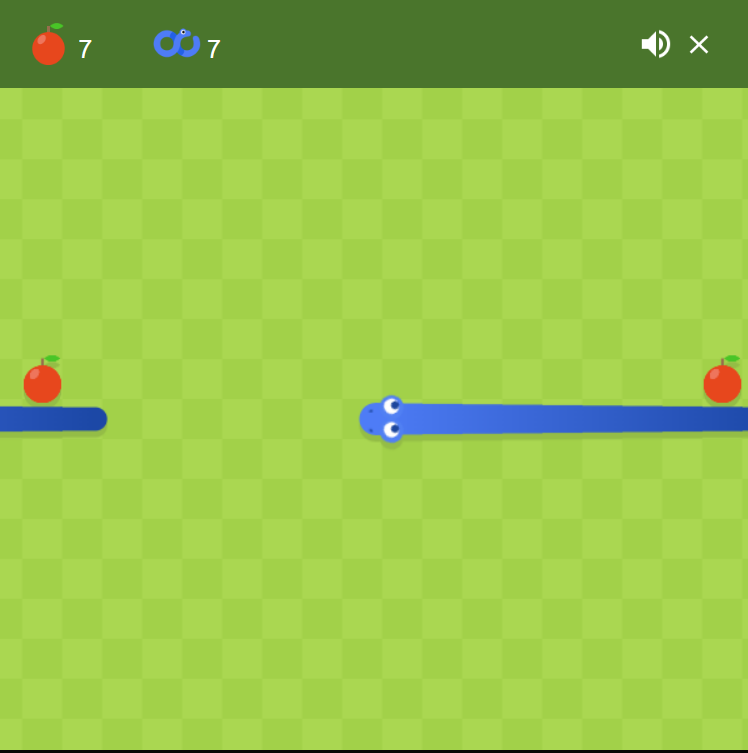
After undertaking this research, it was clear to me that interactive virtual tools to explore topology are basically non-existent. The fact that there is a ⅔ match between the top three search results for interactive and non-interactive topology maths game reinforces this further.

### Current Programs

The top answer to the Quora question also referenced Pacman and Asteroids. These use topological methods as when the player comes off one side of the screen, they enter on the other side. This means that the surface of the game is actually a cylinder or a torus.

Also, google snake uses this method as well:

<https://g.co/kgs/ziM4m1> (20/07/2022)



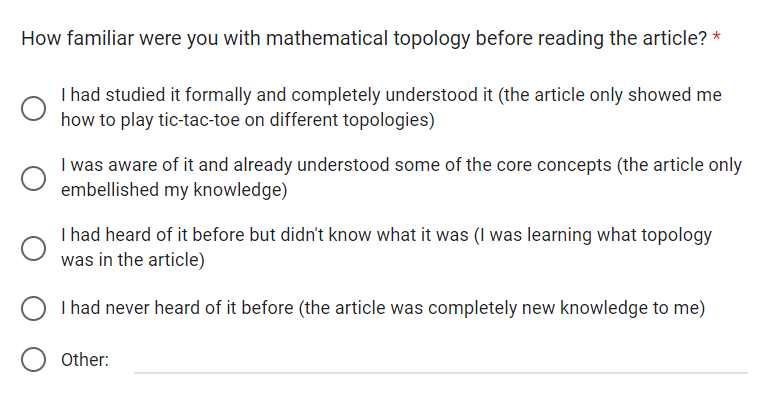
Selecting the infinite snake symbol in the second row of settings sets the game surface to a torus.

This reveals that there are some video games, where the user is playing on a surface with a different topology to that of a plane. However, these games do not stress the use of topology and do not really contain adventurous topologies. Also, they are not really Maths games.

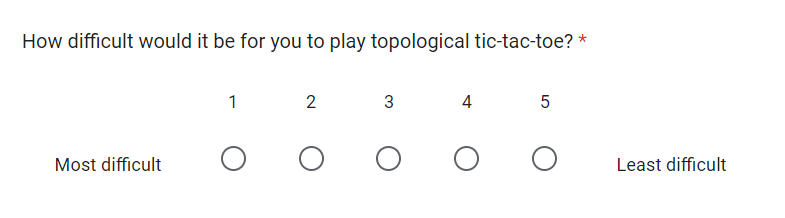
### Questionnaire

The end-users of the game are Mathematicians and Computer Scientists. In order to understand what parameters they would like in the game and what my objectives should be, I created a questionnaire, using Google Forms, to send to three Maths teachers and two Computer Science teachers. The questionnaire contained lots of background and explanatory information and asked them about topology, the game concept, the game interface and the database. I received five responses from 13/07/2022 to 21/07/2022.

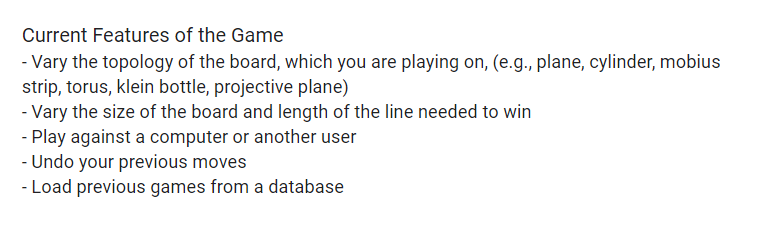
#### Analysis of Responses

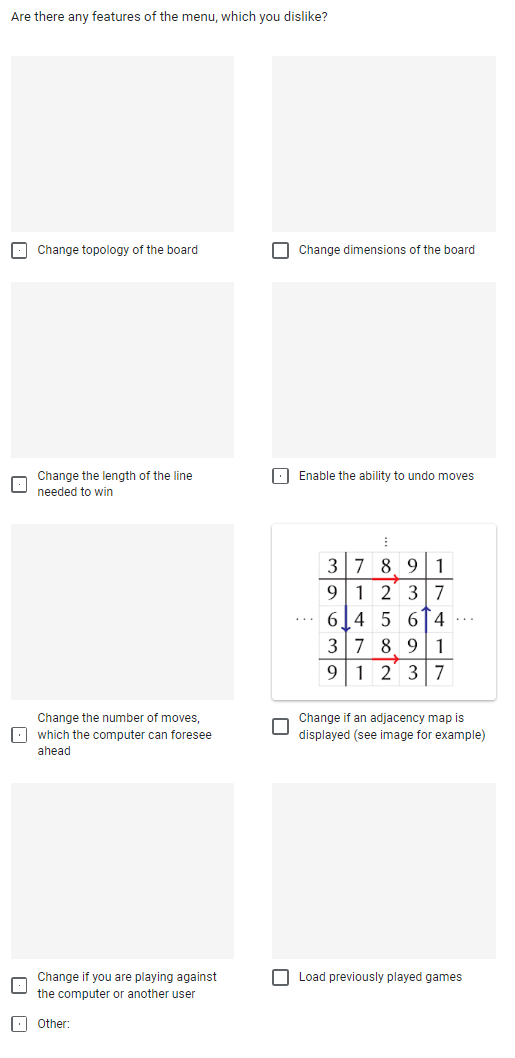


A slim majority (⅗) of the respondents already knew what mathematical topology was. It seemed that the Maths teachers were a lot more familiar with topology than the Computer Science teachers with ⅔ of them having studied it formally before. One of the Computer Science teachers had never heard of topology before. This means that it is important to be beginner friendly with the game (such as including an adjacency map or undo button) if I am making it for Computer Scientists too.



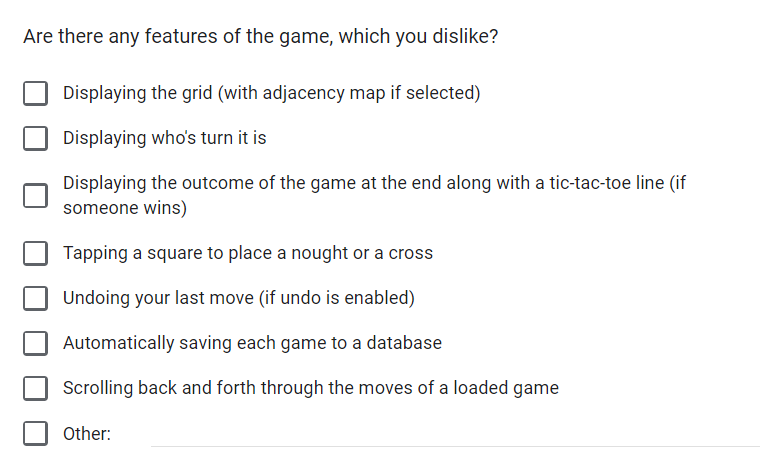
Nobody found it extraordinarily easy or difficult, which indicates that the game has the right balance between challenging you to teach you and being fun and enjoyable. Again, the Computer Science teachers said they would find it marginally harder (they had a mean response of 2.5) to play than the Maths teachers (mean response of 3). This reinforces the findings of the previous question.



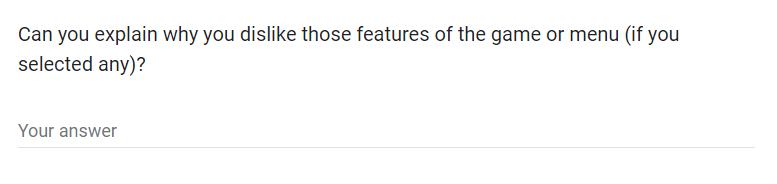


Most respondents (⅗) were happy with all of these features. One respondent said “I feel that allowing a player to undo moves intially [sic] would be good but this function should disappear after a given time”. I think that this would be catered for because whenever the player feels that they are ready to disable the ability to undo moves, they could do that from the menu. Possibly, this respondent misunderstood this function of enabling the ability to undo a given move as opposed to actually undoing a specific move in a game.

Another respondent (say respondent A) said they disliked the ability to change the dimensions of the grid and the length of the line needed to win. I will respond to this in a subsequent question, where they explained why they did not like these features.

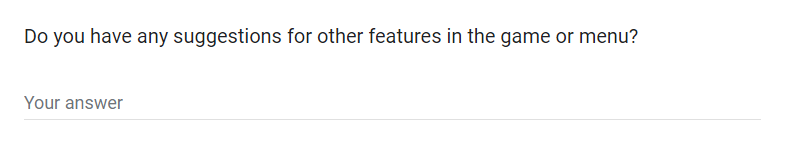


Again, most respondents (⅘) were satisfied with all of the features. However, one (say respondent B) said they did not like automatically saving each game to the database. I will respond to this in a subsequent question where they revealed their reason for this.



Respondent A said their reason was to “keep it simple with size of grid, and length of line”. If they felt that the aforementioned features over-complicate the game, then they would never have to use it. However, it may be wise to only include these as additional objectives as opposed to being core to the completeness of the game. Notwithstanding, if I keep this in mind, when designing the algorithms, then it should be easy to modify them to simply take two extra parameters and not have to change their functionality much.

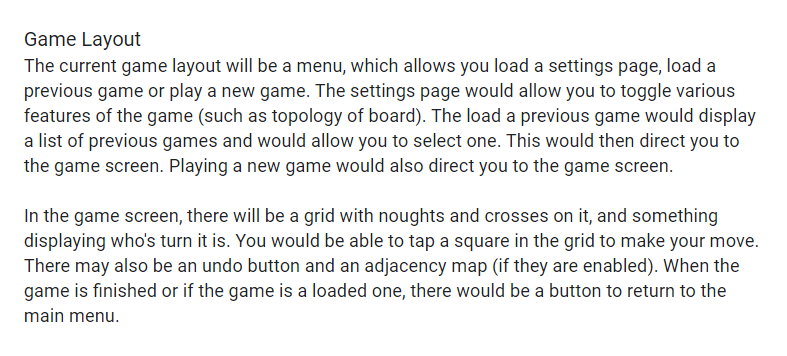
Respondent B said their reason was that “your database could get very big very quickly making your game expensive to host which would then mean the cost gets passed on to users”. If the database is set up properly and normalised (to avoid duplicate data), then I cannot imagine that the size of it would get massively big. And the long query times would only be experienced by users if they select the option to load a previous game.

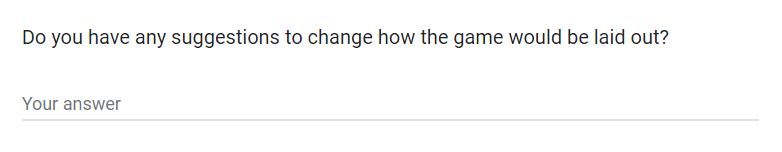


The majority of respondents (⅗) left this question unanswered and one said “I think you are offering quite an extensive range of options already”. This indicates that the game will be sufficiently broad and I do not need to come up with any new features at this stage.

However, the other respondent said “A practice mode where the user is shown potential moves they could make in order to win / advance in the game for the given board” as their first suggestion. I believe this would make the game too big and too much to program in the given time so I will not include it.

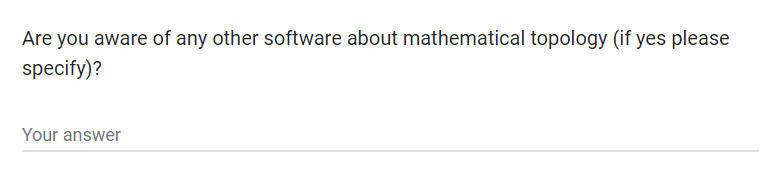
Their second suggestion was “Giving the option to play multiple games, e.g. best of 3, and keep track of who is winning”. I do not believe that this will add too much to the game and so its inclusion is not critical. Also, it is important to note that it will be easy for users to keep track of how many games they have each won themselves. Hence, I will not include it in the game.



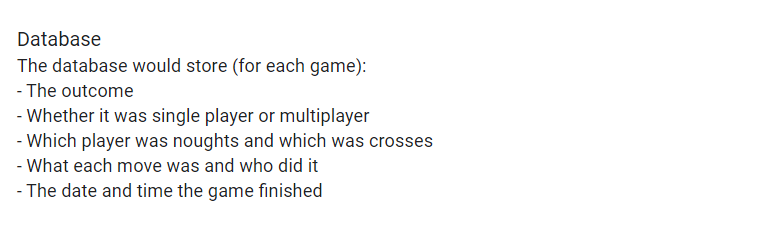


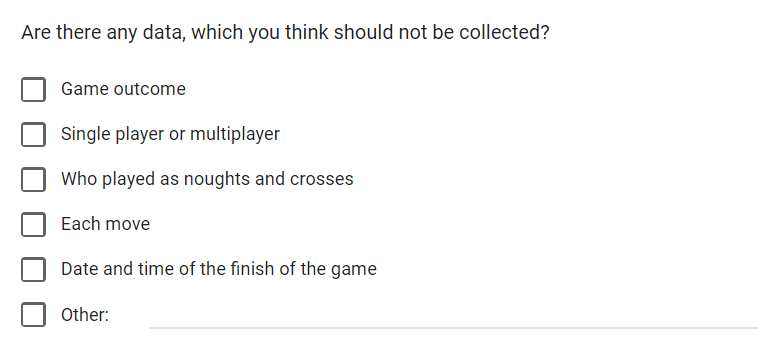
Again, most respondents (⅗) did not have any suggestions. However, the respondent, which suggested playing multiple games, said “scores?”. As I will not be including a rounds or multiple games system, this is not necessary.

Respondent B said “how many loadable games are you going to display (how far back will you go)?”. This is a possible solution for maintaining the size of the database and I will keep it in mind, but I doubt that it will be needed.



⅗ of the respondents were not aware of any other software. One said the 3D Chess game, but this is more of a physical game. The other respondent said “not interactive software ones no. I am aware of websites that suggest activities you can do physically that teach this”. Again this reinforces my earlier research and that my game will be unique.



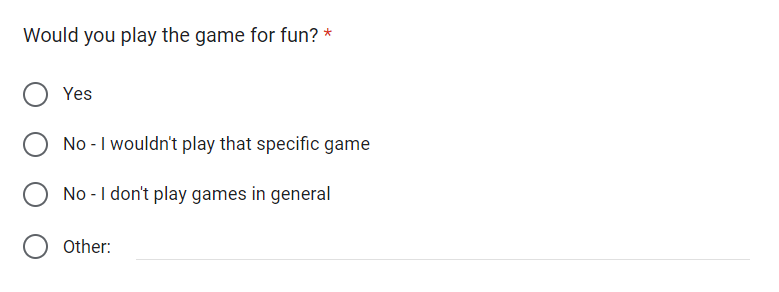


⅘ of the respondents left this question blank, suggesting that I do not need to change the database model. However, respondent B said “who played as noughts and crosses, each move”. I will respond to this in a subsequent question.

It is also important to note that to reproduce each game, some of the game settings must be stored (namely the topology and grid side length) for the purposes of rendering the grid. This felt unnecessarily complex to explain to the prospective users so I left it out.



Respondent B said that they disagreed with each move because it would make the database too large. I must store this data to recreate each previous game. They also said: “who played as which symbol because I feel it is extraneous, if you are going to have a games table with composite key [sic] made of players and store in it the winner you can associate that to a player and not need to store who was what”. I believe that this person has misunderstood what data I will collect. I will not be collecting a username and other information for each player and creating a players table; the database will be specific to that user’s program and only have Player 1, Player 2 and Computer as the possible players. Notwithstanding, Player 1 will always play as crosses, which can be implemented programmatically in the game as opposed to being redundantly stored in the database. Hence, I will not store this data.



⅗ of respondents said “Yes”. One said “Yes, I would have a go” suggesting that they are not too keen on games, but tentative to try this one out. The other respondent said “I think there is a chance I would play it for fun but in certain scenarios, it is more likely I would use it for educational purposes”. This evidences that the game could also be used educationally as well. It is clear to me that this game would be successful in being fun for users to play – the core aim of any game.

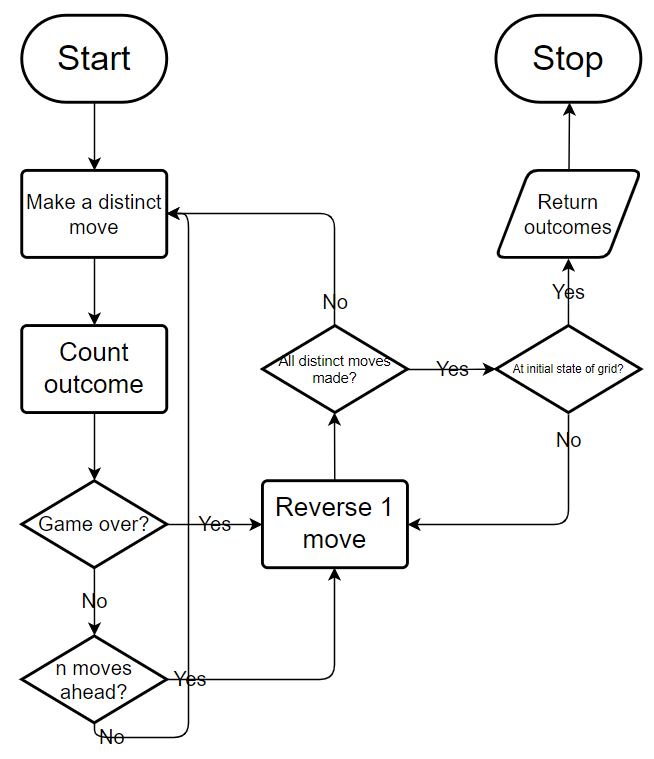
#### Summary

To summarise:

* It is important to make the game beginner friendly so that Computer Scientists can access it too
* Include changing the length of the line needed to win and dimensions of the grid as additional objectives – not core objectives
* Keep the idea of handling different size grids and checking for different lengths of lines in mind when designing the algorithms
* Keep the idea of only storing a set number of games in the database in mind if the database gets too large
* Other software to explore mathematical topology are non-existent making Topological Tic-Tac-Toe unique
* It is unnecessary to store which player played as noughts and crosses as Player 1 always plays as crosses
* Most respondents would play the game for fun with one leaning more towards using it educationally

## Modelling

### Computer Single-Player Algorithm Flowchart



The above flowchart shows an overview of the algorithm the computer will use when deciding which move to take in single-player mode. This will inform the design stage of the overarching structure of the pseudocode algorithm. This algorithm can be implemented as a recursive one as whenever the algorithm gets to the “Make a distinct computer / player move” step, it will have to call itself again from inside itself, parsing the updated grid. n is the recursion depth of the algorithm and, in this context, is the number of moves the computer can foresee ahead. The player will be able to choose n.

### Database Entity Descriptions & ER Diagram

Game(GameID, SettingsID, Outcome, Day, Month, Year, Time)

Move(MoveNo, GameID, Player, Square)

Settings(SettingsID, IsSinglePlayer, Topology, GridSize, LineSize)



The above ER diagram and entity descriptions demonstrate how the database will be set up. Each attribute in the Settings relation could be stored in the Game relation. However, this would mean that (as many games may have the same set of settings) lots of duplicate data would be stored. Hence, there is a separate relation for each unique set of settings. Notwithstanding the advantage, this does mean that a check must be made to ensure that a set of settings has not already been stored in the database when a game has been finished.

Each game has many moves in it but only has one set of settings. Each move is only associated with one game and so only has one set of settings. Each set of settings can be used in many games and hence many moves. Also note that I am including a Player attribute in the Move relation. This is becuase I may decide to allow the user to select which player they want to start. Hence, there are no partial dependencies. This will inform the design stage of what specific SQL queries should be used to create the database tables.

### Data Flow Diagram



The above DFD demonstrates how users playing the game will interact with the database with regards to storing their game data into the database. This will inform the design stage of how the games database should be used when the user completes a game and loads a game.

## Data Volumes

The game is not online so all data stored in the database is specific to each user. The number of records in the game relation will be bounded by how much the user plays the game. So, it will not be too large. The number of records in the settings relation has a maximum value of 216, which again is not too large. The number of records in the move relation will be the largest and a range of 1 to 36 records could be stored per game. Whilst this could get large in multiplicity, a minimalist amount of data is stored per record so it would only be a moderate total amount of data. Also, the data is only accessed when the user chooses to load a game, which is going to be very infrequent. This means that the data velocity is quite low. Overall, the data volume will not be too large with perhaps a moderate amount of data stored in the move relation and the data velocity is very low.

## Proposed Solution Details

### Programming Language

#### Option A

I could create the game in a website format. I would have to use javascript to build the logic and functionality of the game and use HTML to create the interface. I could also use a SQL server or a .db file.

#### Option B

Alternatively, I could create the game using Python 3. In order to implement the graphical user interface, I could either use Python turtle, which is built into Python, or install a third-party module. To implement the database, there are various database modules, which are built into Python.

#### Conclusion

Option A would make the game easier to access for people. However, I doubt that Computer Scientists would have any trouble installing Python 3 and many Mathematicians do a lot of programming so they could easily install Python 3 as well. Also, I have no experience with creating games and implementing databases in javascript and HTML whereas I have done both of the aforementioned in Python before. Hence, I will choose option B.

### Graphical User Interface

#### Option A

Pygame 2 is the most prolific module for creating games in the Python community. There are a lot of online resources available for using it. Also, the games it makes would be efficient and not laggy. However, the user would have to access their command prompt to install Pygame, which could be tricky.

#### Option B

The turtle module is built into Python 3 so would require no further installation. However, it only really exists to help introduce beginners into programming and would not work well in production. This is because it would be laggy and not have many pre written instructions for creating a game.

#### Conclusion

Again, I do not believe that Mathematicians and Computer Scientists will have too much trouble with accessing their command prompt to install Pygame. Also, the turtle module is unsuitable for developing games and I have experience using Pygame so I will choose option A.

### Database

I will use SQLite and a .db file to create the database. This is because SQLite is a built-in module to Python 3 so will not require further installation and I have experience using it before.

## Objectives

### Core Objectives

1. General
   1. The program should be written in Python 3.10.5 with Pygame 2.1.0 as the only 3rd party module
   2. The program should have a clean, user-friendly GUI
   3. The program should be able to run on any machine with Python 3.10.5 and Pygame 2.1.0 installed
2. Menu
   1. There should be a button to:
      1. Direct the user to the settings screen
      2. Direct the user to the load previous game screen
      3. Load a game based on the settings and direct the user to the game screen
3. Settings Screen
   1. The user should be able to:
      1. Select one topology from a list
      2. Enable or disable the ability to undo moves, using a button
      3. Change the number of moves the computer single-player algorithm can foresee ahead (which is actually the recursion depth of the algorithm), using a character input box
      4. Enable or disable the display of an adjacency map, using a button
      5. Change whether the game is single-player or multiplayer
      6. Return to the menu screen, using a button, unless at least one input contains abnormal data
   2. The following topologies should be supported:
      1. Plane
      2. Cylinder
      3. Mobius strip
      4. Torus
   3. The number of moves the computer single-player algorithm can foresee ahead must be between 1 and 5 inclusive
4. Load Previous Game Screen
   1. There should be a list of up to five previous games displaying at any given time
   2. Each item of the list should have:
      1. Which player won or if the game was a draw
      2. Whether the game was single or multiplayer
      3. The topology of the grid in the game
      4. The date and time the game finished
   3. When an item is pressed, it should redirect the user to the game screen with that game loaded
   4. There should be a button to:
      1. Return to the menu
      2. Move to the next list of items (if there are more previous games)
      3. Move to the previous list of items (if the user has moved forward)
   5. The following statistics should be displayed:
      1. Total number of previous games
      2. Win rate of each player
5. Game Screen
   1. The program should display:
      1. A grid with noughts and crosses on it and arrows to denote the (user set) topology
      2. An adjacency map (if enabled)
      3. Who’s turn it is if nobody has won
      4. An undo button (if enabled and the game is not over)
      5. A button to exit to the menu (if the game is over)
   2. At the end of the game, the program should display the outcome and a line (if a player won)
   3. The user should be able to:
      1. Tap a square on the grid to place their nought or cross
      2. Exit back to the menu at the end of the game
   4. Pressing the undo button should undo the last move in multiplayer mode and last two moves in single-player mode (if enabled)
   5. The program should:
      1. Change player turns after each move
      2. Automatically use the computer algorithm to make a move (if the game is single player mode)
      3. End the game if appropriate, checking if it is after each move
   6. When a game ends, the program should save to a database:
      1. The outcome
      2. Each move
      3. The date and time the game finished
      4. The topology of the grid in the game
      5. Whether the game was single or multiplayer
   7. If a previous game has been loaded:
      1. All objectives, with prefix 5, apply with the following exceptions
      2. There should be a forwards and a backwards button to allow the user to go to the next or previous move
      3. There should not be an undo button
      4. The user should not be able to tap a square on the grid
      5. All objectives, with prefix 5.5 and 5.6, do not apply
      6. Objectives 5.1.5 and 5.3.2 apply throughout the game

### Additional Objectives

1. Settings Screen
   1. The user should be able to:
      1. Change the length of the square grid, using a character input box
      2. Change the length of the line needed to win, using a character input box
   2. The following topologies should be supported:
      1. Klein bottle
      2. Projective plane
   3. The following dimensions should be supported:
      1. 3x3
      2. 4x4
      3. 5x5
      4. 6x6
   4. The length of the line needed to win must be greater than zero and less than or equal to the length of the grid
2. Load Previous Game Screen
   1. Each item of the list should have:
      1. The size of the grid in the game
3. Game Screen
   1. When a game ends, the program should save to a database:
      1. The size of the grid in the game
      2. The length of the line needed to win

# Design

## System Summary

### Menu

The user can navigate to the settings, play game or load previous game screen. Navigating to the settings screen and returning from there updates a list of settings. Navigating to the play game screen parses the list of settings to the play game subroutine.

### Settings Screen

The user can change the following settings to do with the grid in a new game:

* Topology
* Side length of square grid
* Length of line needed to win

They can also change:

* Whether there is an undo button in the game screen if they are playing a new game
* The number of moves the computer can foresee ahead (the recursion depth of the computer algorithm)
* Whether there is an adjacency map in the game screen (regardless of if the user is loading a previous game or playing a new game)
* The game mode of a new game

The user can only enter a digit (0 to 9) into: the side length of square grid input; length of line needed to win input or number of moves the computer can foresee ahead input. To validate these inputs:

* For the side length of square grid, the program checks if the digit is one of: 3; 4; 5 or 6
* For the length of line needed to win, the program checks if the digit is non-zero and less than the side length of square grid
* For the number of moves the computer can foresee ahead input, the program checks if the digit is one of: 1; 2; 3; 4 or 5

If any of these checks fail, an error message displays and the return button does not work.

If all user inputs are valid, the user can use a return button to return to the main menu. A list containing the settings will be returned to the main program loop.

### Play Game Screen – Playing a New Game

The program displays:

* A grid
* An adjacency map (if enabled in the list of settings)
* An undo button (if enabled in the list of settings)
* If the game is unfinished, who’s turn it is
* If the game is finished, the outcome of the game and a return button

If the game mode is single-player and it is player 1’s turn, the user can press a square on the grid to place a cross there. The program checks if the game is over. If not, it gets a move from the computer algorithm and places a nought there. Again, the program checks if the game is over. If not, this process is repeated. If an undo button is enabled, the user can press it after the computer has made a move to undo the last two moves.

If the game mode is multiplayer, instead of making a computer move, the user must press a square on the grid to place a nought there. The user can press the undo button at any stage to undo the last move.

If, at any point, the game is over, the program checks if a record for the game settings already exists in the database. If not, it creates a record for the game settings. Then, it saves the game into the game relation (associating the settings id with it). After that, it saves each move in the game to the database. The user can no longer press the grid or the undo button and can only return to the menu.

### Load Previous Game Screen

The program displays a list of up to 5 previous games (with the most recent first), depending on how many total games there are. Each item of the list (a previous game) has a load button and the following information:

* Game outcome
* Game mode (single-player or multiplayer)
* Grid topology
* Grid size
* The date the game finished
* The time the game finished

Pressing a load button directs the user to the play game screen and loads the previous game into that screen.

The program also displays:

* The total number of games
* The percentage of games won by player 1
* The percentage of games won by either player 2 or the computer

If there are more older previous games than the ones currently displayed, the user can press a next button to replace the current list with a list of the next 5 or fewer older previous games.

If there are more newer previous games, than the ones currently displayed, the user can press a previous button to replace the current list with a list of the next 5 newer previous games.

Also, the user can press the return button to return to the main menu.

### Play Game Screen – Loading Previous Game

The program displays:

* A grid
* An adjacency map (if enabled in settings)
* If the game is unfinished, who’s turn it is
* If the game is finished, the outcome of the game
* A return button

If there are more moves in the game, the user can press a next button to update the grid with the next move. If the user is not viewing the first move, they can press a previous button to update the grid to before the current move was made.

At any point, the user can return to the main menu.

### Grid

A move is made by parsing the square (which the player pressed) and the current player to a move method of the grid class. If an adjacency map is enabled, the grid automatically updates each square of the adjacency map, which correspond to each square on the grid due to the grid’s topology.

The grid can also check whether the game is over by checking for a line (whose length is user set) of sequential noughts or crosses. If no line exists and all squares of the grid have been played into the grid check end method returns an indicator that the game is a draw.

When a move is made, the grid displays a circle (for a nought) or an X (for a cross) in that square. If an adjacency map is enabled, it also displays a circle or an X in every corresponding square of the square on the grid. If the game is over, the grid displays a red line through the constituent squares. The line may come of the grid at one end and re-enter the grid at another end due to the grid’s topology.

### Computer Single-Player Algorithm

The recursion depth n of the computer algorithm is the number of moves it simulates ahead of the current state of the board.

The computer algorithm iterates through each 1 move it could make. For each of those moves, it simulates the player and computer making n - 1 moves and then reverses each move and plays a different move. This means that it can simulate all the possible plays the player and computer could make up to n - 1 moves. If, at any point, the game ends, it appends a number associated with the outcome to an outcomes list associated with the first move.

For each of these outcome lists, the computer algorithm generates a score and selects the first move with the highest score.

## Hierarchy Chart



## Classes

### Button

Used to create buttons on any screen in the program.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| x | int | The x-coordinate of the top-left of the button |
| y | int | The y-coordinate of the top-left of the button |
| w | int | The width of the button |
| h | int | The height of the button |
| text | pygame.Surface | The rendered text, which should be in the button |
| colour | tuple (int, int, int) | The rgb background colour of the button |
| state | int | 2 if the mouse is over the button and pressing it or 0 otherwise |
| selected | boolean | Whether the button has been selected |
| selectable | boolean | Whether the button can be selected |
| deselected | boolean | Whether the mouse has been released after pressing the button |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, x, y, w, h, text, selectable=False, is\_small=False | Creates an instance of a button object with the specified inputs |
| update | self, mouse\_x, mouse\_y, mouse\_pressed | Updates the state of the button and if it has been selected based on the position of the mouse and whether or not it is clicking |
| render | self, win | Renders the button on the screen depending on its state and if it is selected |

### List of Buttons

Used to create a list of buttons, of which only one can be selected, on any screen in the program.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| buttons | list of Button, length = button\_count | Each button object in the list of buttons |
| selected | int | The index of the button that has been selected |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, x, y, w, h, button\_count, texts, selected=None | Creates an instance of a list of buttons object with the specified inputs |
| update | self, mouse\_x, mouse\_y, mouse\_pressed | Updates the state of each button based on the position of the mouse and whether or not it is clicking |
| select | self, index | Registers the indexed button as selected within the list and changes its selected attribute to True and changes the previously selected button’s state to False |
| render | self, win | Renders the list of buttons on the screen depending on each button’s state and if it is selected |

### Character Input Box

Used to create two boxes with a prompt in one box on one side and a character input box on the other side on any screen in the game.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| x | int | The x coordinate of the top-left of the box |
| y | int | The y coordinate of the top-left of the box |
| prompt | pygame.Surface | The rendered text to display next to the input box |
| prompt\_w | int | The width of the prompt |
| prompt\_h | int | The height of the prompt |
| input\_text | str | The character, which the user has entered into the input box |
| input | pygame.Surface | The rendered character, which the user has entered into the input box |
| w | int | The width of both boxes |
| h | int | The height of the tallest box |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, x, y, prompt, input\_text | Creates an instance of a character input box object with the specified inputs |
| check\_selected | self, mouse\_x, mouse\_y, mouse\_pressed | Returns true if the user is selecting either box |
| update\_input | self, new\_input\_text | Changes the rendered user character input to the parsed rendered character |
| render | self, win | Renders the character input box to the screen containing the user input |

### Grid

Displays the grid to the game screen of the program, allows the user to place a nought / cross in squares, checks if the game is over and allows the user to cycle through a previous game.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| topology | tuple (int, int) | The topology of the grid, where the first element is 0 if the top and bottom sides are not connected, 1 if they are connected and untwisted and 2 if they are connected and twisted. The second element is the same for the left and right sides |
| grid\_size | int | The number of squares in one side of the grid |
| grid | 2D list  size = (grid\_size + 2) x (grid\_size + 2) | Contains the current state of the grid and the adjacency map around it |
| line\_size | int | The number of squares in the line needed to win |
| is\_adjacency\_map | bool | Whether or not to display and update an adjacency map |
| moves | Stack | A stack containing the previous moves in a new game or all of the moves in a previous game |
| current\_move | int | The index of the move in the moves stack just played |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, topology, grid\_size, line\_size, is\_adjacency\_map, moves=None | Creates an instance of a grid object with the specified inputs |
| get\_state | self, win, mouse\_x, mouse\_y, mouse\_pressed | Returns a tuple containing the row and column of the square if the player is pressing a square or 0 otherwise |
| update\_grid | self, square, value | Updates the parsed square of the grid and (if an adjacency map is enabled) the corresponding squares of the adjacency map with the parsed value |
| corr\_square | self, square | Returns the indices of the real square on the grid, which corresponds to the imaginary square on the adjacency map |
| adj\_square | self, square, direction | Returns the indices of the square, which is adjacent to the parsed square in the parsed direction, on the grid |
| move | self, square, player | Makes the player move by updating the grid and pushing the move onto the moves stack |
| previous\_move | self, is\_interactive | Updates the grid to the previous move in the stack of moves. If a game is being played (so the grid is interactive) it deletes this move |
| next\_move | self, player | Updates the grid to the next move in the stack of moves |
| check\_end | self, square, player | Checks whether the game has ended. Returns: the squares in the winning line if it was won by the parsed player; 3 if the game is a draw or false if no one has won yet |
| add\_squares | self, square, dir\_row, dir\_col, line, player | Adds any squares to the line, which are stemming away from the parsed square in the parsed direction vector |
| check\_line | self, square, player | Checks for a line on the grid stemming from the parsed square and if there is, it returns the squares in the line |
| get\_orientations | self, line | Returns a dictionary with each square in the line as a key and a value of “h”, “v”, “df” or “db” if the line through the square is horizontal, vertical, forward diagonal (like a forward slash) or backward diagonal (like a backward slash) respectively |
| render\_line | self, win, line | Renders the line, that demonstrates which player won, on the grid |
| render | self, win | Renders the grid to the screen |

### Menu

Displays the menu screen and allows the user to interact with the buttons.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| w | int | The width of the screen |
| h | int | The height of the screen |
| settings\_button | Button | The top button on the menu screen, which will redirect the user to the settings screen |
| game\_button | Button | The middle button on the menu screen, which will redirect the user to the game screen |
| previous\_game\_button | Button | The bottom button on the menu screen, which will redirect the user to the load previous game screen |
| mouse\_released | boolean | Whether the mouse has been released from being pressed down |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, w, h | Creates an instance of the menu screen object with the specified inputs |
| loop | self, win, mouse\_x, mouse\_y, mouse\_pressed | Used in the main program loop to update the state of each button and render the menu screen. Returns “Settings” to indicate the settings button has been pressed, “Play Game” for the game button and “Previous Game” for the previous game button |
| render | self, win | Renders the menu screen |

### Previous Game

Displays a previous game screen, containing a list of (up to) five previous games. Allows the user to move to the next or previous list, load a previous game or return to the menu.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| w | int | The width of the previous game screen |
| h | int | The height of the previous game screen |
| return\_button | Button | The button to return to the menu |
| previous\_button | Button | The button to move the user onto the previous list of five previous games, provided that they are not on the first list of (up to) five |
| next\_button | Button | The button to move the user onto the next list of (up to) five previous games, provided that the user is not on the last list of (up to) five |
| mouse\_released | boolean | Whether the mouse has been released from being pressed down |
| total\_games | int | The number of previous games the user has played |
| player1\_win\_rate | float | The percentage of games, which player 1 has won to one decimal place |
| player2\_win\_rate | float | The percentage of games, which either player 2 or the computer has won to one decimal place |
| page | int | The number of the list of five previous games, which the user is currently viewing |
| previous\_games\_info | list of dictionaries, length = 5 | A list of dictionaries containing information on each of the five previous games |
| load\_buttons | list length = 5 | A list of each button object, which will load a previous game |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, w, h | Creates an instance of a previous game screen object |
| load\_previous\_games\_info | self | Populates the previous\_game\_info attribute with the information of the relevant previous games, which are determined by the page attribute |
| next\_page | self | Updates the state of the previous game object to the next list of five previous games |
| previous\_page | self | Updates the state of the previous game object to the previous list of five previous games |
| loop | self, win, mouse\_x, mouse\_y, mouse\_pressed | Used in the main program loop to update the state of each button, move on to the next or previous list of previous games and render the screen. Returns a number representing a GameID to indicate a load button has been pressed and “Return” to indicate the return button has been pressed |
| render\_previous\_game | self, win, num | Renders the “num”th previous game in the previous\_games\_info attribute to the screen |
| render | self, win | Renders the previous game screen |

### Game

Displays the game screen, containing the grid and (if enabled) an adjacency map. If the user is playing a game, it allows the user and another user or the computer algorithm to interact with the grid and (if enabled) undo moves. If the user has loaded a previous game, it allows the user to cycle through the moves.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| w | int | The width of the game screen |
| h | int | The height of the game screen |
| is\_interactive | bool | Whether the grid is interactive (alternatively, if the user is playing or has loaded a game) |
| undo\_enabled | bool | Whether the undo functionality is enabled |
| is\_single\_player | bool | Whether the game mode is single-player |
| return\_button | Button | The button to return to the menu. Only displays if the user is playing a game and has finished the game or if the user has loaded a previous game |
| current\_player | int | The number of the player whose turn it is (2 also corresponds to the computer) |
| outcome | int | Either: the number of the player that has won; 3 if the game is a draw or 0 if the game is unfinished |
| grid | Grid | The grid object, which displays the current state of the game and allows the user(s) / computer to play the game or observe a previous game |
| computer | Computer | The computer algorithm object, which gets the best move for the computer to make against the user (if the user is playing a game and has chosen single-player mode) |
| previous\_button | Button | The button to update the state of the grid to the previous move. Only displays if the user has loaded a game and is not on the first move |
| next\_button | Button | The button to update the state of the grid to the next move. Only displays if the user has loaded a game and is not on the last move |
| undo\_button | Button | The button to undo the last move. Only displays if the user is playing a game, has enabled undo, the game has not ended and they have made a play |
| mouse\_released | boolean | Whether the mouse has been released from being pressed down |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, w, h, settings, moves | Creates an instance of the game screen object with the specified inputs |
| make\_move | self, square | Updates the grid object to register the player’s move, swaps the current player and checks if the game has ended, updating the outcome attribute and calling the save game method accordingly |
| undo\_move | self | Undoes the last move and swaps the current player |
| previous\_move | self | Updates the grid to the previous move, swaps the current player and ensures that the outcome is 0 |
| next\_move | self | Updates the grid to the next move, swaps the current player and updates the outcome attribute |
| make\_computer\_move | self | Gets the next move from the computer algorithm and calls the make move method |
| save\_game | self | If a record containing the game settings does not exist, it inserts one. Inserts records containing the game and the moves in the game into the database |
| loop\_game | self, win, mouse\_x, mouse\_y, mouse\_pressed | If the user is playing a game, it is used in the main program loop to update the state of the grid based on the user input, allow the user to return to the menu and render the game screen. Returns “Return” to indicate the return button has been pressed |
| loop\_previous\_game | self, win, mouse\_x, mouse\_y, mouse\_pressed | If the user has loaded a previous game, it is used in the main program loop to allow the user to cycle through the moves in the game, update the grid accordingly, return to the menu and render the game screen. Returns “Return” to indicate the return button has been pressed |
| render | self, win | Renders the game screen |

### Settings

Displays a settings screen, which allows the user to change: the topology of the grid; the size of the grid; the length of the line needed to win; whether an undo button is displayed; the number of moves the computer algorithm can foresee ahead; whether an adjacency map is displayed and the game mode. Displays an error message if the user enters an invalid input and allows the user to return to the menu if all inputs are valid.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| w | int | The width of the settings screen |
| h | int | The height of the settings screen |
| return\_button | Button | The button to return to the menu |
| topology\_buttons | ListOfButtons | The list of buttons to select the topology of the grid |
| grid\_size\_inp | CharInputBox | The character input box to set the size of the grid |
| line\_size\_inp | CharInputBox | The character input box to set the size of the line needed to win |
| undo\_button | Button | The button to enable or disable the ability to undo a move |
| moves\_inp | CharInputBox | The character input box to set the number of moves the computer algorithm can foresee ahead (its recursion depth) |
| adjacency\_map\_button | Button | The button to enable or disable the display of an adjacency map |
| game\_mode\_buttons | ListOfButtons | The list of buttons to select the game mode |
| mouse\_released | boolean | Whether the mouse has been released from being pressed down |
| error\_msg | str | The multiline string, which should be rendered in the error message box if the user enters a invalid input |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, w, h | Creates an instance of the settings screen object with the specified inputs |
| validate\_inputs | self | Checks if the grid\_size\_inp, line\_size\_inp and moves\_inp attributes are all valid. Updates the error\_msg attribute accordingly |
| loop | self, win, mouse\_x, mouse\_y, mouse\_pressed | Used in the main program loop to update the state of each input object, ensure inputs are valid before the user tries to return to the menu and render the screen. Returns “Grid Size Input” to indicate the grid size character input box has been selected, “Line Size Input” to indicate the line size character input box has been pressed, “Moves Input” to indicated the moves character input box has been pressed and a list containing the settings to indicate the return button has been pressed and what settings to use |
| render\_error | self, win | Renders the error message to the screen |
| render | self, win | Renders the settings screen and the error message if there is one |

### Computer Single-Player Algorithm

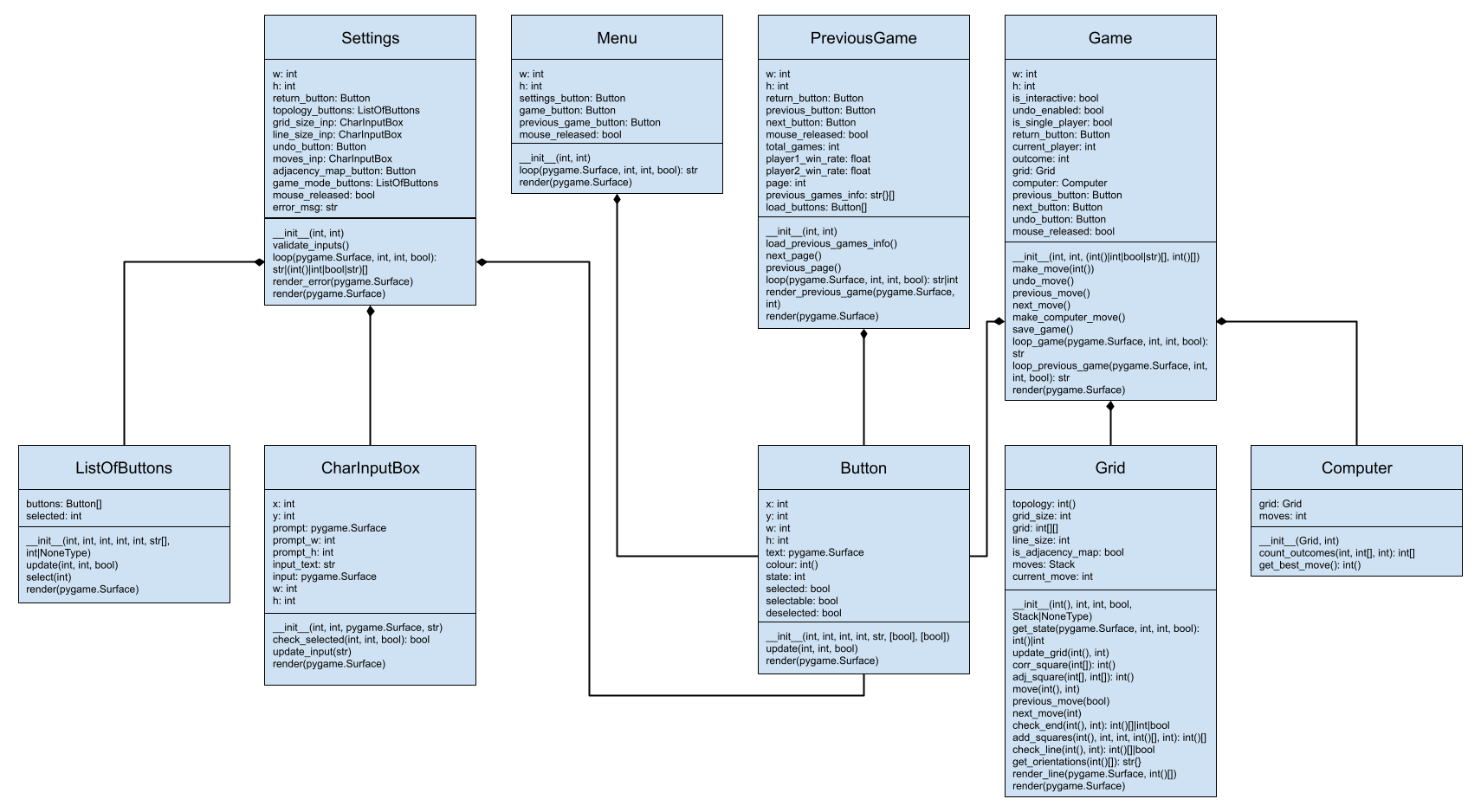
Implements the functionality of the computer single-player algorithm to simulate all possible moves ahead and decide which one is the best.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| grid | Grid | A grid object containing the current state of the grid and all of its settings |
| moves | int | The number of moves, which the computer can simulate ahead from the current state of the grid. It is also the recursion depth of the algorithm |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, grid, moves | Creates an instance of a computer single-player algorithm object with the specified inputs |
| count\_outcomes | self, n, outcomes, player = 1 | Returns a list of outcomes of every possible n moves from the current state of the grid |
| get\_best\_move | self | Obtains a list of outcomes of every possible one move from the current state of the grid and evaluates each list to see which move is the best |

### Class Diagram

Below is a UML class diagram demonstrating how the classes are interrelated in the program.



## Algorithms

### Corresponding Square

This algorithm is used to return the indices of the real square on the grid, which corresponds to the imaginary square on the adjacency map. Where the topology does not allow there to be any correspondence between the grid and adjacency map (such as a plane), the algorithm returns the parsed square.

SUBROUTINE corr\_square(size, topology, square)

corr ← square

FOR dimension ← 0 TO 1

IF topology[dimension] ≠ 0 AND (corr[dimension] = 0 OR corr[dimension] = size + 1) THEN

IF corr[dimension] = 0 THEN

corr[dimension] ← size

ELSE

corr[dimension] ← 1

ENDIF

IF topology[dimension] = -1 THEN

other ← 0 \*\* dimension

corr[other] ← size + 1 - corr[other]

ENDIF

ENDIF

ENDFOR

RETURN corr

ENDSUBROUTINE

### Adjacent Square

This algorithm is used to return the indices of the square, which is adjacent to the parsed square in the parsed direction, on a grid with a specific topology

SUBROUTINE adj\_square(size, topology, square, direction)

adjacent ← [0, 0]

FOR index ← 0 TO 1

adjacent[index] ← square[index] + direction[index]

ENDFOR

RETURN corr\_square(size, topology, adjacent)

ENDSUBROUTINE

### Get Orientation of Line

This algorithm determines whether the parsed squares (which constitute the winning line) are arranged horizontally, vertically, forward diagonally (like a forward slash) or backward diagonally (like a backslash). It uses a dictionary called vect\_to\_string, which maps a direction vector to letter indicating the direction.

SUBROUTINE get\_orientations(line, topology, size)

IF LEN(line) = 1 THEN

RETURN {STR(line[0]): "h"}

ENDIF

line\_orientations ← {}

FOR dir\_row ← -1 TO 1

FOR dir\_col ← -1 TO 1

IF dir\_row = dir\_col = 0 THEN

CONTINUE

ENDIF

next\_square ← line[0]

FOR square\_num ← 0 TO LEN(line) - 1

IF square\_num = 1 THEN

line\_orientations[STR((row, col))] = vect\_to\_string[STR((dir\_row, dir\_col))]

ENDIF

new\_dir\_row, new\_dir\_col = dir\_row, dir\_col

IF square\_num ≠ 0 THEN

IF topology[0] = -1 AND (row = 1 AND dir\_row = -1 OR row = size AND dir\_row = 1) THEN

new\_dir\_col = -new\_dir\_col

ENDIF

IF topology[1] == -1 AND (col == 1 AND dir\_col == -1 OR col == size AND dir\_col == 1) THEN

new\_dir\_row = -new\_dir\_row

ENDIF

ENDIF

row = next\_square[0]

col = next\_square[1]

dir\_row = new\_dir\_row

dir\_col = new\_dir\_col

IF square\_num > 0 THEN

line\_orientations[STR((row, col))] = vect\_to\_string[STR((dir\_row, dir\_col))]

ENDIF

next\_square = corr\_square(size, topology, [row + dir\_row, col + dir\_col])

IF NOT(next\_square IN line) THEN

BREAK

ENDIF

ENDFOR

ENDFOR

ENDFOR

RETURN line\_orientations

ENDSUBROUTINE

### Counting Outcome of Each Move

This algorithm, used as part of the computer algorithm in single-player mode, counts the outcomes of every possible n moves from the current state of the grid. It returns a list of all of the outcomes. The grid parameter is a Grid object.

SUBROUTINE count\_outcomes(grid, player, n, outcomes)

size ← grid.grid\_size

FOR row ← 1 TO size

FOR col ← 1 TO size

IF grid.grid[row][col] = 0 THEN

grid.move([row, col], player)

outcome ← grid.check\_end([row, col], player))

IF TYPE(outcome) = LIST THEN

outcomes.APPEND(player)

ENDIF

IF outcome ≠ FALSE OR n ≤ 1 THEN

grid.previous\_move(TRUE)

ELSE

outcomes ← count\_outcomes(grid, 0 \*\* (player - 1) + 1, n - 1, outcomes)

grid.previous\_move(TRUE)

ENDIF

ENDIF

ENDFOR

ENDFOR

RETURN outcomes

ENDSUBROUTINE

### Get Best Move

This algorithm, used by the computer in single-player mode, calculates the score of each move and returns the move with the highest score. The grid parameter is a Grid object.

SUBROUTINE get\_best\_move(grid, n)

high\_score ← -∞

FOR row ← 1 TO grid.grid\_size

FOR col ← 1 TO grid.grid\_size

IF grid.grid[row][col] = 0 THEN

grid.move([row, col], 2)

outcomes ← count\_outcomes(grid, 1, n, [])

IF LEN(outcomes) = 0 THEN

score ← 0

ELSE

score ← (outcomes.COUNT(2) - outcomes.COUNT(1)) / LEN(outcomes)

ENDIF

IF score > high\_score THEN

high\_score ← score

best\_move ← [row, col]

ENDIF

grid.previous\_move(TRUE)

ENDIF

ENDFOR

ENDFOR

RETURN best\_move

ENDSUBROUTINE

## Data Structures

### List

The list data structure will be used throughout the solution. A list is a mutable, ordered collection of variables with differing data types. For example, a list will be used to store each outcome in the count\_outcomes method of the computer algorithm class. Also, each variable could also be a list making the original list two-dimensional. The grid attribute of the grid class will be implemented as a two-dimensional list as the grid is two-dimensional.

### Dictionary

The dictionary data structure will be used throughout the solution. A dictionary is a mutable one-to-one mapping from a set of keys (of string data type) to a set of values (of any data type). For example, a dictionary will be used to map a string direction vector to a string code (representing the orientation of a line) in the grid class.

### Tuple

The tuple data structure will be used throughout the solution. A tuple is a immutable, ordered collection of variables with differing data types. For example, a tuple will be used to store the topology of the grid in the grid class.

### String

The string data structure will be used throughout the solution. A string is a immutable, ordered collection of characters. For example, a string will be used to store the error message to display to the user in the settings class.

### Stack

The stack data structure will be used during an undo-enabled game to undo the last move. A stack is a LIFO (last in first out) collection of variables with differing data types. When viewing a previous game, to prevent the need to have another list data structure containing the moves and for other reasons, additional functionality to access the item at a given index will be added to the stack. It is noted that this is not usually a feature of a stack.

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| length | int | The maximum number of items the stack can contain |
| data | list | The elements in the stack |

| **Method** | **Parameters** | **Description** |
| --- | --- | --- |
| \_\_init\_\_ | self, length | Creates an instance of a stack object with the parsed parameter |
| push | self, item | Adds an item to the stack |
| pop | self | Returns and removes the last item added to the stack |
| peek | self | Return the last item added to the stack without removing it |
| full | self | Returns whether or not the stack is full |
| empty | self | Returns whether or not the stack is empty |
| index | self, index | Returns the item at the parsed index |

## Database Design

There will need to be a database to store the user’s previous games. The database would need to store each game, the moves in each game and each set of settings. An entity relationship diagram and entity descriptions can be found in the project modelling subsection of the analysis section.

### Entity Descriptions

Game(GameID, SettingsID, Outcome, Day, Month, Year, Time)

| **Field** | **Data Type** | **Description** |
| --- | --- | --- |
| GameID | Integer | Primary key |
| SettingsID | Integer | Foreign key referencing the specific set of settings used in the game |
| Outcome | Integer | The player that won (1 or 2) or 0 if the game was a draw |
| Day | Integer | The day the game finished |
| Month | Integer | The month the game finished |
| Year | Integer | The year the game finished |
| Time | Char(5) | The 24-hour time the game finished in the form “HH:MM” |

Move(MoveNo, GameID, Player, Square)

| **Field** | **Data Type** | **Description** |
| --- | --- | --- |
| MoveNo | Integer | Part of composite primary key |
| GameID | Integer | Part of composite primary key |
| Player | Integer | The player that made the move (1 or 2) |
| Square | Char(3) | The square that the player played in with the form “[row],[col]” |

Settings(SettingsID, IsSinglePlayer, Topology, GridSize, LineSize)

| **Field** | **Data Type** | **Description** |
| --- | --- | --- |
| SettingsID | Integer | Primary key |
| IsSinglePlayer | Integer | Whether the game was single-player (0 = False, 1 = True) |
| Topology | Varchar(16) | The name of the topology of the grid |
| GridSize | Integer | The length of a side of the grid |
| LineSize | Integer | The length of the line needed to win |

## Queries

Below is a sample of queries that are used to create the Game and Move relation, retrieve previous games from the database, write a game to the database and retrieve statistics about the games played. They demonstrate the sophistication of the SQL queries used.

### Create Game Relation

CREATE TABLE Game (

GameID INT PRIMARY KEY AUTOINCREMENT,

SettingsID INT NOT NULL,

Outcome INT NOT NULL,

Day INT NOT NULL,

Month INT NOT NULL,

Year INT NOT NULL,

Time CHAR(5) NOT NULL,

FOREIGN KEY(SettingsID) REFERENCES Settings(SettingID)

);

The above query creates the Game relation. It is set up so that each of the attributes cannot be null apart from the GameID, which allows the GameID to auto-increment with each insertion of a record. Also, it contains a foreign key to a settings record, which was used in the game.

### Create Move Relation

CREATE TABLE Move (

MoveNo INT NOT NULL,

GameID INT NOT NULL,

Player INT NOT NULL,

Square CHAR(3) NOT NULL,

FOREIGN KEY(GameID) REFERENCES Game(GameID),

PRIMARY KEY (MoveNo, GameID)

);

The above query creates the Move relation. It contains a foreign key to a game record, which indicates the game the move was played in. It also contains a composite primary key with both the MoveNo and GameID attributes.

### Insert Record into Game Relation

INSERT INTO Game

VALUES (null, 3, 1, 15, 9, 2022, "11:33");

The above query inserts a previous game into the game table. In this case, the game used the settings set with primary key 3, player one won and it finished on 15/09/2022 at 11:33. The null primary key causes it to be auto-incremented. This query will be parameterised using string interpolation on implementation to allow any game to be inserted into the relation.

### Retrieve Records from Game Relation

SELECT GameID, Outcome, IsSinglePlayer, Topology, GridSize, Day, Month, Year, Time

FROM Game, Settings

WHERE Game.SettingsID = Settings.SettingsID

ORDER BY GameID DESC

LIMIT 5

OFFSET 5

The above query retrieves the 6th to 10th previous game in descending order of finish datetime (as GameID auto-increments). Given that these queries will be stored as strings, the offset can be changed programmatically using string interpolation to retrieve a different set of games. This will need to be done when the user presses the next or back button on the previous games screen.

### Retrieve Records from Setttings Relation

SELECT Topology, GridSize, LineSize, IsSinglePlayer

FROM Settings

WHERE SettingsID = (SELECT SettingsID FROM Game WHERE GameID = 1)

The above query retrieves the settings of the first game in the Game relation. The nested query retrieves the SettingsID of the first game and then this is used to retrieve settings identified by that SettingsID. Given that these queries will be stored as strings, the GameID can be changed programmatically using string interpolation to retrieve the settings for a different game.

### Retrieve Number of Wins for a Player from Game Relation

SELECT COUNT(GameID)

FROM Game

WHERE Outcome = 1

The above query calculates the number of wins for player one, which will be used to calculate their win rate. It does this by using an SQL aggregate function. The WHERE clause can be altered so that the same statistic for player two can be collected.

## UI Diagrams

### Menu



1. This button will direct the user to the settings screen
2. This button will load a new game based on the current settings and direct the user to the game screen
3. This button will load the previous five games from the database and direct the user to the load previous game screen

### Settings Screen



1. This button will save the current settings (returning it to the menu screen) and return the user to the menu as long as all inputs are valid. In this instance, the button will not work as an error message is displaying
2. This list of buttons allows the user to select the topology of the grid they want to play on. The user has selected the cylinder topology
3. This character input box allows the user to input the side length of the grid. The user has inputted 3 or left the input as the default value (which is 3)
4. This character input box allows the user to input the length of the line they need to win the game. The user has inputted 4. This has raised the error as this length must be less than or equal to the side length of the grid (you cannot have four adjacent squares with crosses in them on a 3x3 grid)
5. This button will enable the ability for the user to undo moves in the game. The user has turned this feature off or left the button in its default state (unselected)
6. This character input box allows the user to input the number of moves the computer single-player algorithm can foresee ahead (the recursion depth of the algorithm)
7. This button, when selected, will cause an adjacency map to be displayed around the grid
8. This list of buttons will change whether the game mode is single-player or multiplayer. The user has selected multiplayer
9. This is where any error message will be displayed if the user enters an erroneous input. This specific message is displaying because the user has entered a greater length of line needed to win than grid side length. The user will not be able to return to the menu

### Load Previous Game Screen



1. This button will return the user to the menu
2. This button will redirect the user to the game screen and load that game into the program
3. The games are displayed in descending order of their finish datetime. Finish times are displayed in 24 hour time
4. The winner is displayed as “Player 2” as opposed to “Computer” as the game mode is multiplayer
5. The winner is displayed as “Computer” as opposed to “Player 2” as the game mode is single-player
6. This button will load the list of the previous five previous games (ordered by descending order of finish datetime) into the “Load Previous Game Screen”. It only works if the user is not viewing the most recent (up to) five previous games
7. This button will load the list of the next five previous games (ordered by descending order of finish datetime) into the “Load Previous Game Screen”. It only works if the user is not viewing the oldest (up to) five previous games
8. Statistics for the total number of games and the win rate for player 1 and player 2 / computer are displayed

### Game Screen



1. This button will return the user to the menu
2. This text displays whose turn it currently is or who has won the game. In this instance, the game mode is single-player and player one has won
3. The blue lines, noughts and crosses are the adjacency map. This displays imaginary noughts and crosses so the user can see which noughts and crosses are touching which real squares due to the topology of the grid. In this instance, the user has enabled the “Display Adjacency Map” in the settings
4. If the user had loaded a previous game then the next button would display here unless the user is viewing the final move. Pressing this button would display the next move in the game
5. The black lines, nought and crosses are the real grid. If the game had not ended, then (if it is their go) the user could click a square to place their nought / cross there
6. The red arrows depict the topology of the grid. Two arrows pointing the same direction on parallel sides denote that those sides are touching. Two arrows pointing the opposite direction on parallel sides denote that one of those sides has been rotated 180° about the side’s normal before being joined to the other side. No arrows on parallel sides denote that those sides are not touching. In this instance, the topology is the Klein bottle
7. If the user had loaded a previous game then the previous button would display here unless the user is viewing the first move. This button would display the previous move in the game when pressed
8. The red lines display when one player has won and depict the line of the noughts / crosses, which made the player win. The can also be vertical and diagonal and is only show on the real grid – hence why the line is broken. In this instance, the user must have inputted 3 as the “Length of Line Needed to Win” in the settings
9. If the game has not ended and the user has enabled “Enable Undo” in the settings, then the undo button would display here. This would allow the user to undo the last move made by any player (in multiplayer mode). If the game mode is single-player, after the computer has moved, the user can use this to undo the last two moves

# Technical Solution

## Technical Skills

Code comments highlight examples of technical skills used in the program. This is a compilation of the group A and B models (in the marking grid) used. The file name and number (after each item) index an example of where the model appears.

* Lists (utils.py 9)
* Stacks (grid.py 100)
* Complex mathematical model (grid.py 75)
* Complex user-defined use of object-orientated programming (OOP) model:
  + Classes (grid.py 5)
  + Composition (game.py 63)
  + Interfaces (grid.py 5)
* Simple data model in database (three interlinked tables)
* Multi-dimensional arrays (grid.py 28)
* Dictionaries (grid.py 7)

Also, this is a compilation of the group A algorithms (in the marking grid) used. The file name and number (after each item) index an example of where the algorithm appears.

* Cross-table parameterised SQL (sql.py 19)
* Aggregate SQL functions (sql.py 31)
* List operations (utils.py 9)
* Stack operations (grid.py 100)
* Recursive algorithms (computer.py 6)
* Complex user-defined algorithms (grid.py 125)
* Dynamic generation of objects based on complex user-defined use of OOP model (main.py 21)

## File Structure

Below is the structure, which files and directories have in the program. *Nota bene* that directories are in bold whereas files have normal formatting. There is also a database file.

**Topological Tic-Tac-Toe**

* **MOD**
  + \_\_init\_\_.py
  + computer.py
  + game.py
  + grid.py
  + inputs.py
  + menu.py
  + previous\_game.py
  + settings.py
  + sql.py
  + utils.py
* main.py
* previous\_games.db

## Project Files

All the program code for the game is reproduced below.

### \_\_init\_\_.py

1. from .game import Game
2. from .menu import Menu
3. from .previous\_game import PreviousGame
4. from .settings import Settings
5. from .sql import retrieve\_settings, retrieve\_moves

### computer.py

1. class Computer: # Class
2. def \_\_init\_\_(self, grid, moves):
3. self.grid = grid
4. self.moves = moves
5. def count\_outcomes(
6. self, n, outcomes, player=1
7. ): # Recursive & complex user-defined algorithm
8. size = self.grid.grid\_size
9. for row in range(1, size + 1):
10. for col in range(1, size + 1):
11. if self.grid.grid[row][col] == 0:
12. self.grid.move([row, col], player)
13. outcome = self.grid.check\_end([row, col], player)
14. if type(outcome) == list:
15. outcomes.append(player) # List Operations
16. if outcome is not False or n <= 1:
17. self.grid.previous\_move(True)
18. else:
19. outcomes = self.count\_outcomes(
20. n - 1, outcomes, 0 \*\* (player - 1) + 1
21. )
22. self.grid.previous\_move(True)
23. return outcomes
24. def get\_best\_move(self):
25. high\_score = float("-inf")
26. for row in range(1, self.grid.grid\_size + 1):
27. for col in range(1, self.grid.grid\_size + 1):
28. if self.grid.grid[row][col] == 0:
29. self.grid.move([row, col], 2)
30. outcomes = self.count\_outcomes(self.moves - 1, [])
31. if len(outcomes) == 0:
32. score = 0
33. else:
34. score = (outcomes.count(2) - outcomes.count(1)) / len(outcomes)
35. if score > high\_score:
36. high\_score = score
37. best\_move = (row, col)
38. self.grid.previous\_move(True)
39. return best\_move

### game.py

1. import datetime
2. import pygame
3. from .computer import Computer
4. from .grid import Grid
5. from .inputs import Button
6. from .sql import (
7. retrieve\_settings\_id,
8. insert\_settings,
9. get\_latest\_settings\_id,
10. insert\_game,
11. get\_latest\_game\_id,
12. insert\_move,
13. )
14. from .utils import Stack, format\_time
15. class Game:
16. return\_button\_x\_offset = 0.03
17. return\_button\_y\_offset = 0.05
18. return\_button\_w\_prop = 0.18
19. return\_button\_h\_prop = 0.11
20. title\_font = pygame.font.SysFont("arial", 60, True)
21. title\_y\_offset = 0.03
22. cycle\_button\_x\_offset = 0.1
23. cycle\_button\_w\_prop = 0.07
24. cycle\_button\_h\_prop = 0.05
25. undo\_button\_x\_offset = 0.1
26. undo\_button\_y\_offset = 0.85
27. undo\_button\_w\_prop = 0.15
28. undo\_button\_h\_prop = 0.11
29. topologies = {
30. "(0, 0)": "Plane",
31. "(0, 1)": "Cylinder",
32. "(0, -1)": "Mobius Strip",
33. "(1, 1)": "Torus",
34. "(1, -1)": "Klein Bottle",
35. "(-1, -1)": "Projective Plane",
36. }
37. def \_\_init\_\_(self, w, h, settings, moves):
38. self.w = w
39. self.h = h
40. self.is\_interactive = True if not moves else False
41. self.undo\_enabled = settings[3]
42. self.is\_single\_player = True if settings[6] == "Single-Player" else False
43. self.return\_button = Button(
44. self.return\_button\_x\_offset \* self.w,
45. self.return\_button\_y\_offset \* self.h,
46. self.return\_button\_w\_prop \* self.w,
47. self.return\_button\_h\_prop \* self.h,
48. "Return",
49. )
50. self.current\_player = 1
51. self.outcome = 0
52. if self.is\_interactive:
53. self.grid = Grid(\*(settings[:3] + [settings[5]])) # Composition
54. if self.is\_single\_player:
55. self.computer = Computer(self.grid, settings[4]) # Composition
56. else:
57. cycle\_button\_h = (0.5 - self.cycle\_button\_h\_prop / 2) \* self.h
58. self.previous\_button = Button(
59. self.cycle\_button\_x\_offset \* self.w,
60. cycle\_button\_h,
61. self.cycle\_button\_w\_prop \* self.w,
62. self.cycle\_button\_h\_prop \* self.h,
63. "Previous",
64. False,
65. True,
66. )
67. self.next\_button = Button(
68. (1 - self.cycle\_button\_x\_offset - self.cycle\_button\_w\_prop) \* self.w,
69. cycle\_button\_h,
70. self.cycle\_button\_w\_prop \* self.w,
71. self.cycle\_button\_h\_prop \* self.h,
72. "Next",
73. False,
74. True,
75. )
76. self.grid = Grid(
77. \*(settings[:3] + [settings[5]]), Stack(settings[1] \*\* 2, moves)
78. )
79. if self.undo\_enabled and self.is\_interactive:
80. self.undo\_button = Button(
81. self.undo\_button\_x\_offset \* self.w,
82. self.undo\_button\_y\_offset \* self.h,
83. self.undo\_button\_w\_prop \* self.w,
84. self.undo\_button\_h\_prop \* self.h,
85. "Undo",
86. )
87. self.mouse\_released = False
88. def make\_move(self, square):
89. self.grid.move(square, self.current\_player)
90. outcome = self.grid.check\_end(square, self.current\_player)
91. if outcome:
92. self.outcome = outcome
93. self.save\_game()
94. self.current\_player = 3 - self.current\_player
95. def undo\_move(self):
96. if self.is\_single\_player:
97. for \_ in range(2):
98. self.grid.previous\_move(True)
99. else:
100. self.grid.previous\_move(True)
101. self.current\_player = 3 - self.current\_player
102. def previous\_move(self):
103. self.grid.previous\_move(False)
104. self.outcome = 0
105. self.current\_player = 3 - self.current\_player
106. def next\_move(self):
107. self.grid.next\_move(self.current\_player)
108. final\_move = self.grid.moves.index(self.grid.current\_move)
109. if final\_move == self.grid.moves.peek():
110. self.outcome = self.grid.check\_end(final\_move, self.current\_player)
111. self.current\_player = 3 - self.current\_player
112. def make\_computer\_move(self):
113. square = self.computer.get\_best\_move()
114. self.make\_move(square)
115. def save\_game(self):
116. # Settings Record
117. topology = self.topologies[str(self.grid.topology)]
118. settings\_id = retrieve\_settings\_id(
119. self.is\_single\_player,
120. topology,
121. self.grid.grid\_size,
122. self.grid.line\_size,
123. )
124. if not settings\_id:
125. insert\_settings(
126. self.is\_single\_player,
127. topology,
128. self.grid.grid\_size,
129. self.grid.line\_size,
130. )
131. settings\_id = get\_latest\_settings\_id()
132. # Game Record
133. now = datetime.datetime.now()
134. if not self.outcome == 3:
135. outcome = self.current\_player
136. else:
137. outcome = 0
138. insert\_game(
139. settings\_id,
140. outcome,
141. now.day,
142. now.month,
143. now.year,
144. format\_time(now.hour, now.minute),
145. )
146. # Move Records
147. move\_no = 1
148. game\_id = get\_latest\_game\_id()
149. player = 1
150. final\_move = self.grid.moves.peek()
151. while True:
152. row, col = self.grid.moves.index(move\_no - 1)
153. insert\_move(move\_no, game\_id, player, f"{row},{col}")
154. if (row, col) == final\_move:
155. break
156. move\_no += 1
157. player = 3 - player
158. def loop\_game(self, win, mouse\_x, mouse\_y, mouse\_pressed):
159. self.return\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
160. self.render(win)
161. if not self.outcome:
162. if self.undo\_enabled:
163. self.undo\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
164. grid\_state = self.grid.get\_state(win, mouse\_x, mouse\_y, mouse\_pressed)
165. if self.mouse\_released:
166. if grid\_state and self.grid.grid[grid\_state[0]][grid\_state[1]] == 0:
167. self.make\_move(grid\_state)
168. elif (
169. self.undo\_enabled
170. and self.undo\_button.state == 2
171. and (
172. self.is\_single\_player
173. and self.grid.current\_move >= 1
174. or not self.is\_single\_player
175. and self.grid.current\_move >= 0
176. )
177. ):
178. self.undo\_move()
179. elif self.is\_single\_player and self.current\_player == 2:
180. self.make\_computer\_move()
181. elif self.mouse\_released and self.return\_button.state == 2:
182. return "Return"
183. if self.mouse\_released == mouse\_pressed:
184. self.mouse\_released = not mouse\_pressed
185. def loop\_previous\_game(self, win, mouse\_x, mouse\_y, mouse\_pressed):
186. self.return\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
187. self.previous\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
188. self.next\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
189. self.render(win)
190. if self.mouse\_released:
191. if self.return\_button.state == 2:
192. return "Return"
193. elif self.previous\_button.state == 2 and not self.grid.current\_move == -1:
194. self.previous\_move()
195. elif self.next\_button.state == 2 and not self.outcome:
196. self.next\_move()
197. if self.mouse\_released == mouse\_pressed:
198. self.mouse\_released = not mouse\_pressed
199. def render(self, win):
200. win.fill("white")
201. self.grid.render(win)
202. if not self.is\_interactive or type(self.outcome) == list or self.outcome == 3:
203. self.return\_button.render(win)
204. if type(self.outcome) == list:
205. if self.current\_player == 2:
206. title\_text = "Player 1 Won"
207. else:
208. title\_text = (
209. "Computer" if self.is\_single\_player else "Player 2"
210. ) + " Won"
211. self.grid.render\_line(win, self.outcome)
212. elif self.outcome == 3:
213. title\_text = "Draw"
214. elif self.current\_player == 1:
215. title\_text = "Player 1's Turn"
216. else:
217. title\_text = (
218. "Computer's" if self.is\_single\_player else "Player 2's"
219. ) + " Turn"
220. title = self.title\_font.render(title\_text, True, "black")
221. title\_w = title.get\_size()[0]
222. win.blit(title, (self.w / 2 - title\_w / 2, self.title\_y\_offset \* self.h))
223. if not self.is\_interactive:
224. if not self.grid.current\_move == -1:
225. self.previous\_button.render(win)
226. if not self.outcome:
227. self.next\_button.render(win)
228. elif (
229. self.undo\_enabled and not self.grid.current\_move == -1 and not self.outcome
230. ):
231. self.undo\_button.render(win)

### grid.py

1. import pygame
2. from .utils import Stack
3. class Grid: # Programming to interface
4. grid\_length = 250
5. vect\_to\_string = {
6. "(-1, -1)": "db",
7. "(-1, 0)": "v",
8. "(-1, 1)": "df",
9. "(0, -1)": "h",
10. "(0, 1)": "h",
11. "(1, -1)": "df",
12. "(1, 0)": "v",
13. "(1, 1)": "db",
14. } # Dictionary
15. def \_\_init\_\_(
16. self,
17. topology,
18. grid\_size,
19. line\_size,
20. is\_adjacency\_map,
21. moves=None,
22. ):
23. self.topology = topology
24. self.grid\_size = grid\_size
25. self.grid = [
26. [0 for \_ in range(self.grid\_size + 2)] for \_ in range(self.grid\_size + 2)
27. ] # Multi-dimensional array
28. self.line\_size = line\_size
29. self.is\_adjacency\_map = is\_adjacency\_map
30. self.moves = moves
31. if not self.moves:
32. self.moves = Stack(self.grid\_size\*\*2) # Stack Operations
33. self.current\_move = -1
34. def get\_state(self, win, mouse\_x, mouse\_y, mouse\_pressed):
35. if mouse\_pressed:
36. width, height = win.get\_size()
37. for row in range(self.grid\_size):
38. for col in range(self.grid\_size):
39. if (
40. height / 2
41. - self.grid\_length / 2
42. + self.grid\_length / self.grid\_size \* row
43. < mouse\_y
44. < height / 2
45. - self.grid\_length / 2
46. + self.grid\_length / self.grid\_size \* (row + 1)
47. and width / 2
48. - self.grid\_length / 2
49. + self.grid\_length / self.grid\_size \* col
50. < mouse\_x
51. < width / 2
52. - self.grid\_length / 2
53. + self.grid\_length / self.grid\_size \* (col + 1)
54. ):
55. return (row + 1, col + 1)
56. return 0
57. def update\_grid(self, square, value):
58. self.grid[square[0]][square[1]] = value
59. if self.is\_adjacency\_map:
60. for r, row in enumerate(self.grid):
61. for c, col in enumerate(row):
62. if r in (0, self.grid\_size + 1) or c in (0, self.grid\_size + 1):
63. corr = self.corr\_square([r, c])
64. if corr == square:
65. self.grid[r][c] = value
66. def corr\_square(self, square): # Complex Mathematical Model
67. corr = square[:]
68. for dimension in range(2):
69. if self.topology[dimension] != 0 and (
70. corr[dimension] == 0 or corr[dimension] == self.grid\_size + 1
71. ):
72. if corr[dimension] == 0:
73. corr[dimension] = self.grid\_size
74. else:
75. corr[dimension] = 1
76. if self.topology[dimension] == -1:
77. other = 0\*\*dimension
78. corr[other] = self.grid\_size + 1 - corr[other]
79. return tuple(corr)
80. def adj\_square(self, square, direction):
81. adjacent = [0, 0]
82. for index in range(2):
83. adjacent[index] = square[index] + direction[index]
84. return self.corr\_square(adjacent)
85. def move(self, square, player):
86. self.update\_grid(square, player)
87. self.moves.push(square) # Stack Operations
88. self.current\_move += 1
89. def previous\_move(self, is\_interactive):
90. if is\_interactive:
91. move = self.moves.pop() # Stack Operations
92. else:
93. move = self.moves.index(self.current\_move)
94. self.update\_grid(move, 0)
95. self.current\_move -= 1
96. def next\_move(self, player):
97. self.current\_move += 1
98. move = self.moves.index(self.current\_move)
99. self.update\_grid(move, player)
100. def check\_end(self, square, player): # Programming to interface
101. outcome = self.check\_line(square, player)
102. if outcome:
103. return outcome
104. elif self.moves.full(): # Stack Operations
105. return 3
106. else:
107. return False
108. def add\_squares(
109. self, square, dir\_row, dir\_col, line, player
110. ): # Complex user-defined algorithm
111. row, col = square
112. for \_ in range(self.line\_size - len(line)):
113. new\_row, new\_col = self.adj\_square([row, col], (dir\_row, dir\_col))
114. new\_dir\_row, new\_dir\_col = dir\_row, dir\_col
115. if self.topology[0] == -1 and (
116. row == 1 and dir\_row == -1 or row == self.grid\_size and dir\_row == 1
117. ):
118. new\_dir\_col \*= -1
119. if self.topology[1] == -1 and (
120. col == 1 and dir\_col == -1 or col == self.grid\_size and dir\_col == 1
121. ):
122. new\_dir\_row \*= -1
123. if row == new\_row and col == new\_col:
124. break
125. row, col = new\_row, new\_col
126. dir\_row, dir\_col = new\_dir\_row, new\_dir\_col
127. if row < 1 or row > self.grid\_size or col < 1 or col > self.grid\_size:
128. break
129. elif self.grid[row][col] != player:
130. break
131. else:
132. line.append((row, col))
133. return line
134. def check\_line(self, square, player):
135. line = [square]
136. for dir\_row in range(-1, 1):
137. for dir\_col in range(-1, 2):
138. if dir\_row == dir\_col == 0:
139. break
140. line = self.add\_squares(square, dir\_row, dir\_col, line, player)
141. line = self.add\_squares(square, -dir\_row, -dir\_col, line, player)
142. if len(line) == self.line\_size:
143. return line
144. line = [square]
145. return False
146. def get\_orientations(self, line): # Complex user-defined algorithm
147. if len(line) == 1:
148. return {str(line[0]): "h"}
149. line\_orientations = {}
150. for dir\_row in range(-1, 2):
151. for dir\_col in range(-1, 2):
152. if dir\_row == dir\_col == 0:
153. continue
154. next\_square = line[0]
155. for square\_num in range(len(line)):
156. if square\_num == 1:
157. line\_orientations[str((row, col))] = self.vect\_to\_string[
158. str((dir\_row, dir\_col))
159. ]
160. new\_dir\_row, new\_dir\_col = dir\_row, dir\_col
161. if square\_num != 0:
162. if self.topology[0] == -1 and (
163. row == 1
164. and dir\_row == -1
165. or row == self.grid\_size
166. and dir\_row == 1
167. ):
168. new\_dir\_col \*= -1
169. if self.topology[1] == -1 and (
170. col == 1
171. and dir\_col == -1
172. or col == self.grid\_size
173. and dir\_col == 1
174. ):
175. new\_dir\_row \*= -1
176. row, col = next\_square
177. dir\_row, dir\_col = new\_dir\_row, new\_dir\_col
178. if square\_num > 0:
179. line\_orientations[str((row, col))] = self.vect\_to\_string[
180. str((dir\_row, dir\_col))
181. ]
182. next\_square = self.corr\_square([row + dir\_row, col + dir\_col])
183. if next\_square not in line:
184. break
185. return line\_orientations
186. def render\_line(self, win, line):
187. win\_width, win\_height = win.get\_size()
188. square\_length = self.grid\_length / self.grid\_size
189. left\_x = win\_width / 2 - self.grid\_length / 2
190. top\_y = win\_height / 2 - self.grid\_length / 2
191. line\_orientations = self.get\_orientations(line)
192. for square in line:
193. if line\_orientations[str(square)] == "h":
194. y = top\_y + (square[0] - 0.5) \* square\_length
195. start\_point = (left\_x + (square[1] - 1) \* square\_length, y)
196. end\_point = (left\_x + square[1] \* square\_length, y)
197. elif line\_orientations[str(square)] == "v":
198. x = left\_x + (square[1] - 0.5) \* square\_length
199. start\_point = (x, top\_y + (square[0] - 1) \* square\_length)
200. end\_point = (x, top\_y + square[0] \* square\_length)
201. elif line\_orientations[str(square)] == "df":
202. start\_point = (
203. left\_x + (square[1] - 1) \* square\_length,
204. top\_y + square[0] \* square\_length,
205. )
206. end\_point = (
207. left\_x + square[1] \* square\_length,
208. top\_y + (square[0] - 1) \* square\_length,
209. )
210. else:
211. start\_point = (
212. left\_x + (square[1] - 1) \* square\_length,
213. top\_y + (square[0] - 1) \* square\_length,
214. )
215. end\_point = (
216. left\_x + square[1] \* square\_length,
217. top\_y + square[0] \* square\_length,
218. )
219. pygame.draw.line(win, "red", start\_point, end\_point)
220. def render(self, win):
221. win\_width, win\_height = win.get\_size()
222. square\_length = self.grid\_length / self.grid\_size
223. left\_x = win\_width / 2 - self.grid\_length / 2
224. right\_x = win\_width / 2 + self.grid\_length / 2
225. top\_y = win\_height / 2 - self.grid\_length / 2
226. bottom\_y = win\_height / 2 + self.grid\_length / 2
227. topology\_line\_dim = 20
228. piece\_size = square\_length \* 0.3
229. # Vertical Grid Lines
230. for line\_num in range(self.grid\_size + 1):
231. line\_x = left\_x + line\_num \* square\_length
232. if self.is\_adjacency\_map:
233. pygame.draw.line(
234. win,
235. "blue",
236. (line\_x, top\_y - square\_length),
237. (line\_x, bottom\_y + square\_length),
238. )
239. pygame.draw.line(win, "black", (line\_x, top\_y), (line\_x, bottom\_y))
240. # Horizontal Grid Lines
241. for line\_num in range(self.grid\_size + 1):
242. line\_y = top\_y + line\_num \* square\_length
243. if self.is\_adjacency\_map:
244. pygame.draw.line(
245. win,
246. "blue",
247. (left\_x - square\_length, line\_y),
248. (right\_x + square\_length, line\_y),
249. )
250. pygame.draw.line(win, "black", (left\_x, line\_y), (right\_x, line\_y))
251. # Top & Bottom Topology Arrows
252. if self.topology[0] != 0:
253. end\_pos = (win\_width / 2 + topology\_line\_dim / 2, top\_y)
254. pygame.draw.line(
255. win,
256. "red",
257. (win\_width / 2 - topology\_line\_dim / 2, top\_y - topology\_line\_dim),
258. end\_pos,
259. )
260. pygame.draw.line(
261. win,
262. "red",
263. (win\_width / 2 - topology\_line\_dim / 2, top\_y + topology\_line\_dim),
264. end\_pos,
265. )
266. if self.topology[0] == 1:
267. end\_pos = (win\_width / 2 + topology\_line\_dim / 2, bottom\_y)
268. pygame.draw.line(
269. win,
270. "red",
271. (
272. win\_width / 2 - topology\_line\_dim / 2,
273. bottom\_y - topology\_line\_dim,
274. ),
275. end\_pos,
276. )
277. pygame.draw.line(
278. win,
279. "red",
280. (
281. win\_width / 2 - topology\_line\_dim / 2,
282. bottom\_y + topology\_line\_dim,
283. ),
284. end\_pos,
285. )
286. else:
287. end\_pos = (win\_width / 2 - topology\_line\_dim / 2, bottom\_y)
288. pygame.draw.line(
289. win,
290. "red",
291. (
292. win\_width / 2 + topology\_line\_dim / 2,
293. bottom\_y - topology\_line\_dim,
294. ),
295. end\_pos,
296. )
297. pygame.draw.line(
298. win,
299. "red",
300. (
301. win\_width / 2 + topology\_line\_dim / 2,
302. bottom\_y + topology\_line\_dim,
303. ),
304. end\_pos,
305. )
306. # Left & Right Topology Arrows
307. if self.topology[1] != 0:
308. end\_pos = (left\_x, win\_height / 2 - topology\_line\_dim / 2)
309. pygame.draw.line(
310. win,
311. "red",
312. (left\_x - topology\_line\_dim, win\_height / 2 + topology\_line\_dim / 2),
313. end\_pos,
314. )
315. pygame.draw.line(
316. win,
317. "red",
318. (left\_x + topology\_line\_dim, win\_height / 2 + topology\_line\_dim / 2),
319. end\_pos,
320. )
321. if self.topology[1] == 1:
322. end\_pos = (right\_x, win\_height / 2 - topology\_line\_dim / 2)
323. pygame.draw.line(
324. win,
325. "red",
326. (
327. right\_x - topology\_line\_dim,
328. win\_height / 2 + topology\_line\_dim / 2,
329. ),
330. end\_pos,
331. )
332. pygame.draw.line(
333. win,
334. "red",
335. (
336. right\_x + topology\_line\_dim,
337. win\_height / 2 + topology\_line\_dim / 2,
338. ),
339. end\_pos,
340. )
341. else:
342. end\_pos = (right\_x, win\_height / 2 + topology\_line\_dim / 2)
343. pygame.draw.line(
344. win,
345. "red",
346. (
347. right\_x - topology\_line\_dim,
348. win\_height / 2 - topology\_line\_dim / 2,
349. ),
350. end\_pos,
351. )
352. pygame.draw.line(
353. win,
354. "red",
355. (
356. right\_x + topology\_line\_dim,
357. win\_height / 2 - topology\_line\_dim / 2,
358. ),
359. end\_pos,
360. )
361. # Noughts & Crosses
362. for r, row in enumerate(self.grid):
363. for c, square in enumerate(row):
364. if self.grid[r][c] == 0:
365. continue
366. centre = (
367. left\_x + (c - 0.5) \* square\_length,
368. top\_y + (r - 0.5) \* square\_length,
369. )
370. if 1 <= r <= self.grid\_size and 1 <= c <= self.grid\_size:
371. colour = "black"
372. else:
373. colour = "blue"
374. if self.grid[r][c] == 2:
375. pygame.draw.circle(win, colour, centre, piece\_size, 1)
376. else:
377. pygame.draw.line(
378. win,
379. colour,
380. (centre[0] - piece\_size, centre[1] - piece\_size),
381. (centre[0] + piece\_size, centre[1] + piece\_size),
382. )
383. pygame.draw.line(
384. win,
385. colour,
386. (centre[0] - piece\_size, centre[1] + piece\_size),
387. (centre[0] + piece\_size, centre[1] - piece\_size),
388. )

### inputs.py

1. import pygame
2. from .utils import point\_in\_rect
3. pygame.init()
4. pygame.font.init()
5. class Button:
6. button\_font = pygame.font.SysFont("arial", 50, True)
7. small\_button\_font = pygame.font.SysFont("arial", 13)
8. offset\_x = 5
9. offset\_y = 5
10. def \_\_init\_\_(self, x, y, w, h, text, selectable=False, is\_small=False):
11. self.x = x
12. self.y = y
13. self.w = w
14. self.h = h
15. if is\_small:
16. self.text = self.small\_button\_font.render(text, True, "black")
17. else:
18. self.text = self.button\_font.render(text, True, "black")
19. self.colour = (150, 150, 150)
20. self.state = 0
21. self.selected = False
22. self.selectable = selectable
23. if self.selectable:
24. self.deselected = True
25. def update(self, mouse\_x, mouse\_y, mouse\_pressed):
26. if (
27. point\_in\_rect(mouse\_x, mouse\_y, self.x, self.y, self.w, self.h)
28. and mouse\_pressed
29. ):
30. self.state = 2
31. if self.selectable and self.deselected:
32. self.selected = bool(1 - int(self.selected))
33. self.deselected = False
34. else:
35. self.state = 0
36. if self.selectable and not self.deselected:
37. self.deselected = True
38. def render(self, win):
39. if self.state == 2 or self.selected:
40. colour = tuple([max(colour\_comp - 50, 0) for colour\_comp in self.colour])
41. else:
42. colour = self.colour
43. pygame.draw.rect(win, colour, (self.x, self.y, self.w, self.h))
44. win.blit(self.text, (self.x + self.offset\_x, self.y + self.offset\_y))
45. class ListOfButtons:
46. offset = 5
47. def \_\_init\_\_(self, x, y, w, h, button\_count, texts, selected=None):
48. self.buttons = []
49. for i in range(button\_count):
50. self.buttons.append(Button(x, y, w, h, texts[i], False, True))
51. x += self.buttons[-1].w + self.offset
52. self.selected = selected
53. if type(self.selected) == int:
54. self.select(self.selected)
55. def update(self, mouse\_x, mouse\_y, mouse\_pressed):
56. for i, button in enumerate(self.buttons):
57. button.update(mouse\_x, mouse\_y, mouse\_pressed)
58. if button.state == 2:
59. self.select(i)
60. def select(self, index):
61. if type(self.selected) == int:
62. self.buttons[self.selected].selected = False
63. self.buttons[index].selected = True
64. self.selected = index
65. def render(self, win):
66. for button in self.buttons:
67. button.render(win)
68. class CharInputBox:
69. offset\_x = 5
70. offset\_y = 5
71. input\_font = pygame.font.SysFont("arial", 30, True)
72. def \_\_init\_\_(self, x, y, prompt, input\_text):
73. self.x = x
74. self.y = y
75. self.prompt = prompt
76. self.prompt\_w, self.prompt\_h = self.prompt.get\_size()
77. self.input\_text = input\_text
78. self.input = self.input\_font.render(self.input\_text, True, "black")
79. input\_w, input\_h = self.input.get\_size()
80. self.w = self.prompt\_w + input\_w + self.offset\_x \* 4
81. self.h = max(self.prompt\_h, input\_h) + self.offset\_y \* 2
82. def check\_selected(self, mouse\_x, mouse\_y, mouse\_pressed):
83. if (
84. point\_in\_rect(mouse\_x, mouse\_y, self.x, self.y, self.w, self.h)
85. and mouse\_pressed
86. ):
87. return True
88. def update\_input(self, new\_input\_text):
89. self.input\_text = new\_input\_text
90. self.input = self.input\_font.render(self.input\_text, True, "black")
91. def render(self, win):
92. pygame.draw.rect(win, "black", (self.x, self.y, self.w, self.h), 1)
93. win.blit(self.prompt, (self.x + self.offset\_x, self.y + self.offset\_y))
94. win.blit(
95. self.input,
96. (self.x + self.prompt\_w + self.offset\_x \* 3, self.y + self.offset\_y),
97. )
98. line\_x = self.x + self.prompt\_w + self.offset\_x \* 2
99. pygame.draw.line(win, "black", (line\_x, self.y), (line\_x, self.y + self.h))

### menu.py

1. import pygame
2. from .inputs import Button
3. pygame.init()
4. pygame.font.init()
5. class Menu:
6. title = pygame.font.SysFont("arial", 75, True).render("Menu", True, "black")
7. title\_w = title.get\_size()[0]
8. title\_offset = 0.05
9. button\_w\_prop = 0.6
10. button\_h\_prop = 0.17
11. button\_top\_offset = 0.3
12. button\_y\_offset = 0.05
13. def \_\_init\_\_(self, w, h):
14. self.w = w
15. self.h = h
16. button\_x = self.w / 2 - self.button\_w\_prop \* self.w / 2
17. button\_w = self.button\_w\_prop \* self.w
18. button\_h = self.button\_h\_prop \* self.h
19. self.settings\_button = Button(
20. button\_x,
21. self.button\_top\_offset \* self.h,
22. button\_w,
23. button\_h,
24. "Settings",
25. )
26. self.game\_button = Button(
27. button\_x,
28. self.settings\_button.y
29. + self.settings\_button.h
30. + self.button\_y\_offset \* self.h,
31. button\_w,
32. button\_h,
33. "Play Game",
34. )
35. self.previous\_game\_button = Button(
36. button\_x,
37. self.game\_button.y + self.game\_button.h + self.button\_y\_offset \* self.h,
38. button\_w,
39. button\_h,
40. "Load Previous Game",
41. )
42. self.mouse\_released = False
43. def loop(self, win, mouse\_x, mouse\_y, mouse\_pressed):
44. self.settings\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
45. self.game\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
46. self.previous\_game\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
47. self.render(win)
48. if self.mouse\_released:
49. if self.settings\_button.state == 2:
50. return "Settings"
51. if self.game\_button.state == 2:
52. return "Play Game"
53. if self.previous\_game\_button.state == 2:
54. return "Previous Game"
55. if self.mouse\_released == mouse\_pressed:
56. self.mouse\_released = not mouse\_pressed
57. def render(self, win):
58. win.fill("white")
59. win.blit(
60. self.title, (self.w / 2 - self.title\_w / 2, self.title\_offset \* self.h)
61. )
62. self.settings\_button.render(win)
63. self.game\_button.render(win)
64. self.previous\_game\_button.render(win)

### previous\_game.py

1. import math
2. import pygame
3. from .inputs import Button
4. from .sql import retrieve\_previous\_games, retrieve\_win\_rates, calculate\_total\_games
5. class PreviousGame:
6. return\_button\_x\_offset = 0.05
7. return\_button\_y\_offset = 0.05
8. return\_button\_w\_prop = 0.18
9. return\_button\_h\_prop = 0.11
10. title = pygame.font.SysFont("arial", 50, True).render(
11. "Load Previous Game", True, "black"
12. )
13. title\_w = title.get\_size()[0]
14. title\_y\_offset = 0.05
15. load\_button\_x\_offset = 0.1
16. load\_button\_y\_offset = 0.18
17. load\_button\_w\_prop = 0.14
18. load\_button\_h\_prop = 0.11
19. load\_button\_y\_space = 0.03
20. info\_stats\_font = pygame.font.SysFont("arial", 13)
21. info\_col1\_x\_offset = 0.03
22. info\_col2\_x\_offset = 0.3
23. move\_button\_x\_offset = 0.05
24. move\_button\_y\_offset = 0.87
25. move\_button\_w\_prop = 0.23
26. move\_button\_h\_prop = 0.11
27. stats\_font = pygame.font.SysFont("arial", 15)
28. stats\_y\_offset = 0.87
29. def \_\_init\_\_(self, w, h):
30. self.w = w
31. self.h = h
32. self.return\_button = Button(
33. self.return\_button\_x\_offset \* self.w,
34. self.return\_button\_y\_offset \* self.h,
35. self.return\_button\_w\_prop \* self.w,
36. self.return\_button\_h\_prop \* self.h,
37. "Return",
38. )
39. self.previous\_button = Button(
40. self.move\_button\_x\_offset \* self.w,
41. self.move\_button\_y\_offset \* self.h,
42. self.move\_button\_w\_prop \* self.w,
43. self.move\_button\_h\_prop \* self.h,
44. "Previous",
45. )
46. self.next\_button = Button(
47. (1 - self.move\_button\_x\_offset - self.move\_button\_w\_prop) \* self.w,
48. self.move\_button\_y\_offset \* self.h,
49. self.move\_button\_w\_prop \* self.w,
50. self.move\_button\_h\_prop \* self.h,
51. "Next",
52. )
53. self.mouse\_released = False
54. self.total\_games = calculate\_total\_games()
55. self.player1\_win\_rate, self.player2\_win\_rate = retrieve\_win\_rates()
56. self.page = 1
57. self.previous\_games\_info = [{} for \_ in range(5)]
58. self.load\_previous\_games\_info()
59. self.load\_buttons = []
60. for count in range(5):
61. self.load\_buttons.append(
62. Button(
63. self.load\_button\_x\_offset \* self.w,
64. (
65. self.load\_button\_y\_offset
66. + (self.load\_button\_h\_prop + self.load\_button\_y\_space) \* count
67. )
68. \* self.h,
69. self.load\_button\_w\_prop \* self.w,
70. self.load\_button\_h\_prop \* self.h,
71. "Load",
72. )
73. )
74. def load\_previous\_games\_info(self):
75. games\_infos = retrieve\_previous\_games((self.page - 1) \* 5)
76. for i in range(5):
77. if i < len(games\_infos):
78. game\_info = games\_infos[i]
79. previous\_game\_info = self.previous\_games\_info[i]
80. previous\_game\_info["Game ID"] = game\_info[0]
81. if game\_info[2]:
82. previous\_game\_info["Game Mode"] = "Single-Player"
83. else:
84. previous\_game\_info["Game Mode"] = "Multiplayer"
85. if game\_info[1] == 1:
86. previous\_game\_info["Game Outcome"] = "Player 1 Won"
87. elif game\_info[1] == 0:
88. previous\_game\_info["Game Outcome"] = "Draw"
89. elif previous\_game\_info["Game Mode"] == "Single-Player":
90. previous\_game\_info["Game Outcome"] = "Computer Won"
91. else:
92. previous\_game\_info["Game Outcome"] = "Player 2 Won"
93. previous\_game\_info["Topology"] = game\_info[3]
94. previous\_game\_info["Grid Size"] = f"{game\_info[4]}x{game\_info[4]}"
95. previous\_game\_info[
96. "Date"
97. ] = f"{game\_info[5]}/{game\_info[6]}/{game\_info[7]}"
98. previous\_game\_info["Time"] = game\_info[8]
99. else:
100. self.previous\_games\_info[i] = {}
101. def next\_page(self):
102. self.page += 1
103. self.load\_previous\_games\_info()
104. def previous\_page(self):
105. self.page -= 1
106. self.load\_previous\_games\_info()
107. def loop(self, win, mouse\_x, mouse\_y, mouse\_pressed):
108. self.return\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
109. self.previous\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
110. self.next\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
111. for load\_button in self.load\_buttons:
112. load\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
113. self.render(win)
114. if self.mouse\_released:
115. if self.return\_button.state == 2:
116. return "Return"
117. if self.previous\_button.state == 2 and self.page > 1:
118. self.previous\_page()
119. elif self.next\_button.state == 2 and self.page < math.ceil(
120. self.total\_games / 5
121. ):
122. self.next\_page()
123. for i, load\_button in enumerate(self.load\_buttons):
124. if load\_button.state == 2 and self.previous\_games\_info[i] != {}:
125. return self.previous\_games\_info[i]["Game ID"]
126. if self.mouse\_released == mouse\_pressed:
127. self.mouse\_released = not mouse\_pressed
128. def render\_previous\_game(self, win, num):
129. load\_button = self.load\_buttons[num]
130. previous\_game\_info = self.previous\_games\_info[num]
131. load\_button.render(win)
132. y = (
133. self.load\_button\_y\_offset
134. + (self.load\_button\_h\_prop + self.load\_button\_y\_space) \* num
135. ) \* self.h
136. info\_col1 = [
137. f"Game Outcome: {previous\_game\_info['Game Outcome']}",
138. f"Game Mode: {previous\_game\_info['Game Mode']}",
139. f"Topology: {previous\_game\_info['Topology']}",
140. f"Grid Size: {previous\_game\_info['Grid Size']}",
141. ]
142. y\_offset = 0
143. for info\_col1\_line in info\_col1:
144. info\_col1\_rendered\_line = self.info\_stats\_font.render(
145. info\_col1\_line, True, "black"
146. )
147. win.blit(
148. info\_col1\_rendered\_line,
149. (
150. (
151. self.load\_button\_x\_offset
152. + self.load\_button\_w\_prop
153. + self.info\_col1\_x\_offset
154. )
155. \* self.w,
156. y + y\_offset,
157. ),
158. )
159. y\_offset += info\_col1\_rendered\_line.get\_size()[1]
160. info\_col2 = [
161. f"Date: {previous\_game\_info['Date']}",
162. f"Time: {previous\_game\_info['Time']}",
163. ]
164. y\_offset = 0
165. for info\_col2\_line in info\_col2:
166. info\_col2\_rendered\_line = self.info\_stats\_font.render(
167. info\_col2\_line, True, "black"
168. )
169. win.blit(
170. info\_col2\_rendered\_line,
171. (
172. (
173. self.load\_button\_x\_offset
174. + self.load\_button\_w\_prop
175. + self.info\_col2\_x\_offset
176. )
177. \* self.w,
178. y + y\_offset,
179. ),
180. )
181. y\_offset += info\_col2\_rendered\_line.get\_size()[1]
182. def render(self, win):
183. win.fill("white")
184. self.return\_button.render(win)
185. win.blit(
186. self.title,
187. (self.w / 2 - self.title\_w / 2, self.title\_y\_offset \* self.h),
188. )
189. for i in range(5):
190. if self.previous\_games\_info[i]:
191. self.render\_previous\_game(win, i)
192. self.previous\_button.render(win)
193. self.next\_button.render(win)
194. stats\_text = [
195. f"Total Games: {self.total\_games}",
196. f"Player 1 Win Rate: {round(self.player1\_win\_rate \* 100, 1)}%",
197. f"Player 2 / Computer Win Rate: {round(self.player2\_win\_rate \* 100, 1)}%",
198. ]
199. y\_offset = 0
200. for stats\_text\_line in stats\_text:
201. stats\_line = self.stats\_font.render(stats\_text\_line, True, "black")
202. stats\_line\_w, stats\_line\_h = stats\_line.get\_size()
203. win.blit(
204. stats\_line,
205. (
206. self.w / 2 - stats\_line\_w / 2,
207. self.stats\_y\_offset \* self.h + y\_offset,
208. ),
209. )
210. y\_offset += stats\_line\_h

### settings.py

1. import pygame
2. from .inputs import Button, ListOfButtons, CharInputBox
3. class Settings:
4. return\_button\_x\_offset = 0.05
5. return\_button\_y\_offset = 0.05
6. return\_button\_w\_prop = 0.18
7. return\_button\_h\_prop = 0.11
8. title = pygame.font.SysFont("arial", 75, True).render("Settings", True, "black")
9. title\_w = title.get\_size()[0]
10. title\_y\_offset = 0.05
11. error\_msg\_font = pygame.font.SysFont("arial", 18)
12. error\_msg\_x\_offset = 0.7
13. error\_msg\_y\_offset = 0.05
14. error\_msg\_w\_prop = 0.25
15. error\_msg\_h\_prop = 0.15
16. input\_x\_offset = 0.1
17. input\_x\_spacing = 0.05
18. input\_y\_spacing = 0.04
19. topology\_prompt = pygame.font.SysFont("arial", 15, True).render(
20. "Select Topology", True, "black"
21. )
22. topology\_prompt\_w, topology\_prompt\_h = topology\_prompt.get\_size()
23. topology\_buttons\_y\_offset = 0.25
24. topology\_button\_w\_prop = 0.11
25. topology\_button\_h\_prop = 0.05
26. grid\_size\_prompt = pygame.font.SysFont("arial", 15, True).render(
27. "Side Length of Square Grid", True, "black"
28. )
29. line\_size\_prompt = pygame.font.SysFont("arial", 15, True).render(
30. "Length of Line Needed to Win", True, "black"
31. )
32. undo\_button\_w\_prop = 0.09
33. undo\_button\_h\_prop = 0.05
34. moves\_prompt = pygame.font.SysFont("arial", 10, True).render(
35. "Number of Moves Computer can Foresee Ahead", True, "black"
36. )
37. adjacency\_map\_button\_w\_prop = 0.15
38. adjacency\_map\_button\_h\_prop = 0.05
39. game\_mode\_prompt = pygame.font.SysFont("arial", 20, True).render(
40. "Game Mode", True, "black"
41. )
42. game\_mode\_prompt\_x, game\_mode\_prompt\_h = game\_mode\_prompt.get\_size()
43. game\_mode\_button\_w\_prop = 0.09
44. game\_mode\_button\_h\_prop = 0.05
45. topologies = ((0, 0), (0, 1), (0, -1), (1, 1), (1, -1), (-1, -1))
46. game\_modes = ("Single-Player", "Multiplayer")
47. def \_\_init\_\_(self, w, h):
48. self.w = w
49. self.h = h
50. self.return\_button = Button(
51. self.return\_button\_x\_offset \* self.w,
52. self.return\_button\_y\_offset \* self.h,
53. self.return\_button\_w\_prop \* self.w,
54. self.return\_button\_h\_prop \* self.h,
55. "Return",
56. )
57. x = self.input\_x\_offset \* self.w
58. self.topology\_buttons = ListOfButtons(
59. x + self.topology\_prompt\_w + self.input\_x\_spacing \* self.w,
60. self.topology\_buttons\_y\_offset \* self.h,
61. self.topology\_button\_w\_prop \* self.w,
62. self.topology\_button\_h\_prop \* self.h,
63. 6,
64. [
65. "Plane",
66. "Cylinder",
67. "Mobius Strip",
68. "Torus",
69. "Klein Bottle",
70. "Projective Plane",
71. ],
72. 0,
73. )
74. self.grid\_size\_inp = CharInputBox(
75. x,
76. (
77. self.topology\_buttons\_y\_offset
78. + self.topology\_button\_h\_prop
79. + self.input\_y\_spacing
80. )
81. \* self.h,
82. self.grid\_size\_prompt,
83. "3",
84. )
85. self.line\_size\_inp = CharInputBox(
86. x,
87. self.grid\_size\_inp.y + self.grid\_size\_inp.h + self.input\_y\_spacing \* self.h,
88. self.line\_size\_prompt,
89. "3",
90. )
91. self.undo\_button = Button(
92. x,
93. self.line\_size\_inp.y + self.line\_size\_inp.h + self.input\_y\_spacing \* self.h,
94. self.undo\_button\_w\_prop \* self.w,
95. self.undo\_button\_h\_prop \* self.h,
96. "Enable Undo",
97. True,
98. True,
99. )
100. self.moves\_inp = CharInputBox(
101. x,
102. self.undo\_button.y + self.undo\_button.h + self.input\_y\_spacing \* self.h,
103. self.moves\_prompt,
104. "3",
105. )
106. self.adjacency\_map\_button = Button(
107. x,
108. self.moves\_inp.y + self.moves\_inp.h + self.input\_y\_spacing \* self.h,
109. self.adjacency\_map\_button\_w\_prop \* self.w,
110. self.adjacency\_map\_button\_h\_prop \* self.h,
111. "Display Adjacency Map",
112. True,
113. True,
114. )
115. self.game\_mode\_buttons = ListOfButtons(
116. x + self.game\_mode\_prompt\_x + self.input\_x\_spacing \* self.w,
117. self.adjacency\_map\_button.y
118. + self.adjacency\_map\_button.h
119. + self.input\_y\_spacing \* self.h,
120. self.game\_mode\_button\_w\_prop \* self.w,
121. self.game\_mode\_button\_h\_prop \* self.h,
122. 2,
123. ["Single Player", "Multi-Player"],
124. 0,
125. )
126. self.mouse\_released = False
127. self.error\_msg = None
128. def validate\_inputs(self):
129. grid\_size = int(self.grid\_size\_inp.input\_text)
130. line\_size = int(self.line\_size\_inp.input\_text)
131. moves = int(self.moves\_inp.input\_text)
132. if grid\_size not in (3, 4, 5, 6):
133. self.error\_msg = "Side Length of Square\nGrid must be 3, 4, 5 or 6"
134. elif not line\_size > 0:
135. self.error\_msg = "Length of Line Needed\nto Win must be greater than 0"
136. elif not line\_size <= grid\_size:
137. self.error\_msg = "Length of Line Needed to\nWin cannot be greater than\nSide Length of Square Grid"
138. elif not moves > 0:
139. self.error\_msg = (
140. "Number of Moves\nComputer can Foresee\nAhead must be greater\nthan 0"
141. )
142. elif not moves <= 5:
143. self.error\_msg = (
144. "Number of Moves\nComputer can Foresee\nAhead cannot be more\nthan 5"
145. )
146. else:
147. self.error\_msg = ""
148. def loop(self, win, mouse\_x, mouse\_y, mouse\_pressed):
149. self.return\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
150. self.topology\_buttons.update(mouse\_x, mouse\_y, mouse\_pressed)
151. self.undo\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
152. self.adjacency\_map\_button.update(mouse\_x, mouse\_y, mouse\_pressed)
153. self.game\_mode\_buttons.update(mouse\_x, mouse\_y, mouse\_pressed)
154. self.validate\_inputs()
155. self.render(win)
156. if self.mouse\_released:
157. if self.grid\_size\_inp.check\_selected(mouse\_x, mouse\_y, mouse\_pressed):
158. return "Grid Size Input"
159. if self.line\_size\_inp.check\_selected(mouse\_x, mouse\_y, mouse\_pressed):
160. return "Line Size Input"
161. if self.moves\_inp.check\_selected(mouse\_x, mouse\_y, mouse\_pressed):
162. return "Moves Input"
163. if not self.error\_msg and self.return\_button.state == 2:
164. return [
165. self.topologies[self.topology\_buttons.selected],
166. int(self.grid\_size\_inp.input\_text),
167. int(self.line\_size\_inp.input\_text),
168. self.undo\_button.selected,
169. int(self.moves\_inp.input\_text),
170. self.adjacency\_map\_button.selected,
171. self.game\_modes[self.game\_mode\_buttons.selected],
172. ]
173. if self.mouse\_released == mouse\_pressed:
174. self.mouse\_released = not mouse\_pressed
175. def render\_error(self, win):
176. pygame.draw.rect(
177. win,
178. "red",
179. (
180. self.error\_msg\_x\_offset \* self.w,
181. self.error\_msg\_y\_offset \* self.h,
182. self.error\_msg\_w\_prop \* self.w,
183. self.error\_msg\_h\_prop \* self.h,
184. ),
185. )
186. error\_msg\_lines = self.error\_msg.split("\n")
187. y\_offset = 3
188. for error\_msg\_line in error\_msg\_lines:
189. error\_line = self.error\_msg\_font.render(error\_msg\_line, True, "black")
190. error\_line\_w, error\_line\_h = error\_line.get\_size()
191. win.blit(
192. error\_line,
193. (
194. (self.error\_msg\_x\_offset + self.error\_msg\_w\_prop / 2) \* self.w
195. - error\_line\_w / 2,
196. self.error\_msg\_y\_offset \* self.h + y\_offset,
197. ),
198. )
199. y\_offset += error\_line\_h
200. def render(self, win):
201. win.fill("white")
202. self.return\_button.render(win)
203. win.blit(
204. self.title,
205. (self.w / 2 - self.title\_w / 2, self.title\_y\_offset \* self.h),
206. )
207. if self.error\_msg:
208. self.render\_error(win)
209. win.blit(
210. self.topology\_prompt,
211. (self.input\_x\_offset \* self.w, self.topology\_buttons\_y\_offset \* self.h),
212. )
213. win.blit(
214. self.game\_mode\_prompt,
215. (self.input\_x\_offset \* self.w, self.game\_mode\_buttons.buttons[0].y),
216. )
217. self.topology\_buttons.render(win)
218. self.grid\_size\_inp.render(win)
219. self.line\_size\_inp.render(win)
220. self.undo\_button.render(win)
221. self.moves\_inp.render(win)
222. self.adjacency\_map\_button.render(win)
223. self.game\_mode\_buttons.render(win)

### sql.py

1. import sqlite3
2. def sql\_connection(func):
3. def wrapper(\*args, \*\*kwargs):
4. connection = sqlite3.connect("previous\_games.db")
5. cursor = connection.cursor()
6. rv = func(cursor, \*args, \*\*kwargs)
7. cursor.close()
8. connection.commit()
9. connection.close()
10. return rv
11. return wrapper
12. @sql\_connection
13. def retrieve\_previous\_games(cursor, offset): # Cross-table parameterised SQL
14. query = f"""SELECT GameID, Outcome, IsSinglePlayer, Topology, GridSize, Day, Month, Year, Time
15. FROM Game, Settings
16. WHERE Game.SettingsID = Settings.SettingsID
17. ORDER BY GameID DESC
18. LIMIT 5
19. OFFSET {offset};"""
20. cursor.execute(query)
21. return cursor.fetchall()
22. @sql\_connection
23. def calculate\_total\_games(cursor): # Aggregate SQL functions
24. query = """SELECT COUNT(GameID)
25. FROM Game;"""
26. cursor.execute(query)
27. return cursor.fetchall()[0][0]
28. @sql\_connection
29. def retrieve\_win\_rates(cursor):
30. player1\_query = """SELECT COUNT(GameID)
31. FROM Game
32. WHERE Outcome = 1;"""
33. cursor.execute(player1\_query)
34. player1\_wins = cursor.fetchall()[0][0]
35. player2\_query = """SELECT COUNT(GameID)
36. FROM Game
37. WHERE Outcome = 2;"""
38. cursor.execute(player2\_query)
39. player2\_wins = cursor.fetchall()[0][0]
40. total\_games = calculate\_total\_games()
41. return player1\_wins / total\_games, player2\_wins / total\_games
42. @sql\_connection
43. def retrieve\_settings\_id(cursor, is\_single\_player, topology, grid\_size, line\_size):
44. query = f"""SELECT SettingsID
45. FROM Settings
46. WHERE IsSinglePlayer = {is\_single\_player}
47. AND Topology = \"{topology}\"
48. AND GridSize = {grid\_size}
49. AND LineSize = {line\_size};"""
50. cursor.execute(query)
51. result = cursor.fetchall()
52. if bool(result):
53. return result[0][0]
54. return False
55. @sql\_connection
56. def insert\_settings(cursor, is\_single\_player, topology, grid\_size, line\_size):
57. query = f"""INSERT INTO Settings
58. VALUES (NULL, {is\_single\_player}, \"{topology}\", {grid\_size}, {line\_size});"""
59. cursor.execute(query)
60. @sql\_connection
61. def get\_latest\_settings\_id(cursor):
62. query = f"""SELECT SettingsID
63. FROM Settings
64. ORDER BY SettingsID DESC
65. LIMIT 1"""
66. cursor.execute(query)
67. return cursor.fetchall()[0][0]
68. @sql\_connection
69. def insert\_game(cursor, settings\_id, outcome, day, month, year, time):
70. query = f"""INSERT INTO Game
71. VALUES (NULL, {settings\_id}, {outcome}, {day}, {month}, {year}, \"{time}\");"""
72. cursor.execute(query)
73. @sql\_connection
74. def get\_latest\_game\_id(cursor):
75. query = f"""SELECT GameID
76. FROM Game
77. ORDER BY GameID DESC
78. LIMIT 1"""
79. cursor.execute(query)
80. return cursor.fetchall()[0][0]
81. @sql\_connection
82. def insert\_move(cursor, move\_no, game\_id, player, square):
83. query = f"""INSERT INTO Move
84. VALUES ({move\_no}, {game\_id}, {player}, \"{square}\")"""
85. cursor.execute(query)
86. @sql\_connection
87. def retrieve\_settings(cursor, game\_id): # Cross-table parameterised SQL
88. query = f"""SELECT Topology, GridSize, LineSize, IsSinglePlayer
89. FROM Settings
90. WHERE SettingsID = (SELECT SettingsID FROM Game WHERE GameID = {game\_id});"""
91. cursor.execute(query)
92. return cursor.fetchall()[0]
93. @sql\_connection
94. def retrieve\_moves(cursor, game\_id):
95. query = f"""SELECT Square
96. FROM Move
97. WHERE GameID = {game\_id}
98. ORDER BY MoveNo;"""
99. cursor.execute(query)
100. return cursor.fetchall()

### utils.py

1. class Stack:
2. def \_\_init\_\_(self, length, data=None):
3. self.length = length
4. self.data = data
5. if self.data == None:
6. self.data = []
7. def push(self, item):
8. self.data.append(item) # List Operations
9. def pop(self):
10. return self.data.pop() # List Operations
11. def peek(self):
12. return self.data[-1] # List Operations
13. def full(self):
14. if len(self.data) == self.length: # List Operations
15. return True
16. return False
17. def empty(self):
18. if len(self.data) == 0: # List Operations
19. return True
20. return False
21. def index(self, index):
22. return self.data[index] # List Operations
23. def point\_in\_rect(point\_x, point\_y, rect\_x, rect\_y, rect\_w, rect\_h):
24. if rect\_x <= point\_x <= rect\_x + rect\_w and rect\_y <= point\_y <= rect\_y + rect\_h:
25. return True
26. return False
27. def round\_sf(value, sf):
28. digits = 0
29. temp\_value = value
30. while temp\_value >= 1:
31. temp\_value //= 10
32. digits += 1
33. return round(value / 10\*\*digits, sf) \* 10\*\*digits
34. def format\_time(hour, minute):
35. hour\_string = str(hour) if hour >= 10 else f"0{hour}"
36. minute\_string = str(minute) if minute >= 10 else f"0{minute}"
37. return f"{hour\_string}:{minute\_string}"

### main.py

1. from MOD import \*
2. import pygame
3. pygame.init()
4. W, H = 800, 600
5. WIN = pygame.display.set\_mode((W, H))
6. CLOCK = pygame.time.Clock()
7. FPS = 40
8. TOPOLOGIES = {
9. "Plane": (0, 0),
10. "Cylinder": (0, 1),
11. "Mobius Strip": (0, -1),
12. "Torus": (1, 1),
13. "Klein Bottle": (1, -1),
14. "Projective Plane": (-1, -1),
15. }
16. def run\_menu():
17. menu = Menu(W, H) # Dynamic generation of objects
18. outcome = None
19. while not outcome:
20. for event in pygame.event.get():
21. if event.type == pygame.QUIT:
22. return "Quit"
23. mouse\_x, mouse\_y = pygame.mouse.get\_pos()
24. outcome = menu.loop(WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])
25. pygame.display.update()
26. CLOCK.tick(FPS)
27. return outcome
28. def run\_previous\_game():
29. previous\_game = PreviousGame(W, H) # Dynamic generation of objects
30. outcome = None
31. while not outcome:
32. for event in pygame.event.get():
33. if event.type == pygame.QUIT:
34. return "Quit"
35. mouse\_x, mouse\_y = pygame.mouse.get\_pos()
36. outcome = previous\_game.loop(
37. WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0]
38. )
39. pygame.display.update()
40. CLOCK.tick(FPS)
41. return outcome
42. def run\_game(settings, moves=None):
43. game = Game(W, H, settings, moves) # Dynamic generation of objects
44. outcome = None
45. while not outcome:
46. for event in pygame.event.get():
47. if event.type == pygame.QUIT:
48. return "Quit"
49. mouse\_x, mouse\_y = pygame.mouse.get\_pos()
50. if moves:
51. outcome = game.loop\_previous\_game(
52. WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0]
53. )
54. else:
55. outcome = game.loop\_game(
56. WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0]
57. )
58. pygame.display.update()
59. CLOCK.tick(FPS)
60. return outcome
61. def run\_settings():
62. settings = Settings(W, H)
63. outcome = None
64. char\_inp = None
65. while not outcome:
66. for event in pygame.event.get():
67. if event.type == pygame.QUIT:
68. return "Quit"
69. elif char\_inp and event.type == pygame.KEYDOWN:
70. keys = pygame.key.get\_pressed()
71. if keys[pygame.K\_0]:
72. char = "0"
73. elif keys[pygame.K\_1]:
74. char = "1"
75. elif keys[pygame.K\_2]:
76. char = "2"
77. elif keys[pygame.K\_3]:
78. char = "3"
79. elif keys[pygame.K\_4]:
80. char = "4"
81. elif keys[pygame.K\_5]:
82. char = "5"
83. elif keys[pygame.K\_6]:
84. char = "6"
85. elif keys[pygame.K\_7]:
86. char = "7"
87. elif keys[pygame.K\_8]:
88. char = "8"
89. elif keys[pygame.K\_9]:
90. char = "9"
91. else:
92. continue
93. if char\_inp == "Grid Size Input":
94. settings.grid\_size\_inp.update\_input(char)
95. elif char\_inp == "Line Size Input":
96. settings.line\_size\_inp.update\_input(char)
97. else:
98. settings.moves\_inp.update\_input(char)
99. char\_inp = None
100. mouse\_x, mouse\_y = pygame.mouse.get\_pos()
101. outcome = settings.loop(WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])
102. if outcome and "Input" in outcome:
103. char\_inp = outcome
104. outcome = None
105. pygame.display.update()
106. CLOCK.tick(FPS)
107. return outcome
108. def main():
109. settings = [(0, 0), 3, 3, False, 3, False, "Single-Player"]
110. while True:
111. outcome = run\_menu()
112. if outcome == "Settings":
113. outcome = run\_settings()
114. settings = outcome
115. elif outcome == "Play Game":
116. outcome = run\_game(settings)
117. elif outcome == "Previous Game":
118. outcome = run\_previous\_game()
119. if type(outcome) == int:
120. topology, grid\_size, line\_size, is\_single\_player = retrieve\_settings(
121. outcome
122. )
123. topology = TOPOLOGIES[topology]
124. outcome = run\_game(
125. [
126. topology,
127. grid\_size,
128. line\_size,
129. False,
130. settings[4],
131. settings[5],
132. "Single-Player" if is\_single\_player else "Multiplayer",
133. ],
134. [
135. (int(move[0][0]), int(move[0][2]))
136. for move in retrieve\_moves(outcome)
137. ],
138. )
139. if outcome == "Quit":
140. return
141. if \_\_name\_\_ == "\_\_main\_\_":
142. main()
143. pygame.quit()

# Testing

## Test Database

To test loading previous games and cycling through them, I will create a test database. I will need to show how the games were carried out and their settings so I know what data must be stored.

### Previous Games

#### Game 1

##### Settings

Game Mode: Single-Player

Topology: Plane

Grid Size: 3

Line Size: 3

##### Moves

The game ends in a draw.

| **Move #** | **Player** | **Grid & Adjacency Map** |
| --- | --- | --- |
| 0 | N/A |  |
| 1 | 1 |  |
| 2 | Computer |  |
| 3 | 1 |  |
| 4 | Computer |  |
| 5 | 1 |  |
| 6 | Computer |  |
| 7 | 1 |  |
| 8 | Computer |  |
| 9 | 1 |  |

#### Game 2

##### Settings

Game Mode: Multiplayer

Topology: Torus

Grid Size: 5

Line Size: 2

##### Moves

Player 2 wins.

| **Move #** | **Player** | **Grid & Adjacency Map** |
| --- | --- | --- |
| 0 | N/A |  |
| 1 | 1 |  |
| 2 | 2 |  |
| 3 | 1 |  |
| 4 | 2 |  |

#### Game 3

##### Settings

Game Mode: Single-Player

Topology: Projective Plane

Grid Size: 6

Line Size: 5

##### Moves

Player 1 wins.

| **Move #** | **Player** | **Grid & Adjacency Map** |
| --- | --- | --- |
| 0 | N/A |  |
| 1 | 1 |  |
| 2 | Computer |  |
| 3 | 1 |  |
| 4 | Computer |  |
| 5 | 1 |  |
| 6 | Computer |  |
| 7 | 1 |  |
| 8 | Computer |  |
| 9 | 1 |  |

### Database

This database contains all the data to accurately reproduce the above three games. It has records of their settings as well.

#### Game

| **GameID** | **SettingsID** | **Outcome** | **Day** | **Month** | **Year** | **Time** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 16 | 11 | 2022 | “20:30” |
| 2 | 2 | 2 | 25 | 12 | 2022 | “12:00” |
| 3 | 3 | 1 | 1 | 1 | 2023 | “00:00” |

#### Move

| **MoveNo** | **GameID** | **Player** | **Square** |
| --- | --- | --- | --- |
| 1 | 1 | 1 | “2,2” |
| 2 | 1 | 2 | “1,1” |
| 3 | 1 | 1 | “3,3” |
| 4 | 1 | 2 | “3,1” |
| 5 | 1 | 1 | “2,1” |
| 6 | 1 | 2 | “2,3” |
| 7 | 1 | 1 | “3,2” |
| 8 | 1 | 2 | “1,2” |
| 9 | 1 | 1 | “1,3” |
| 1 | 2 | 1 | “4,4” |
| 2 | 2 | 2 | “1,1” |
| 3 | 2 | 1 | “5,2” |
| 4 | 2 | 2 | “5,5” |
| 1 | 3 | 1 | “1,1” |
| 2 | 3 | 2 | “3,3” |
| 3 | 3 | 1 | “5,6” |
| 4 | 3 | 2 | “4,3” |
| 5 | 3 | 1 | “4,5” |
| 6 | 3 | 2 | “5,3” |
| 7 | 3 | 1 | “3,4” |
| 8 | 3 | 2 | “6,3” |
| 9 | 3 | 1 | “2,3” |

#### Settings

| **SettingsID** | **IsSinglePlayer** | **Topology** | **GridSize** | **LineSize** |
| --- | --- | --- | --- | --- |
| 1 | 1 | “Plane” | 3 | 3 |
| 2 | 0 | “Torus” | 5 | 2 |
| 3 | 1 | “Projective Plane” | 6 | 5 |

### Queries

The following queries were used to insert the data into the database.

#### Game

INSERT INTO Game (SettingsID, Outcome, Day, Month, Year, Time)

VALUES

(1, 0, 16, 11, 2022, "20:30"),

(2, 2, 25, 12, 2022, "12:00"),

(3, 1, 1, 1, 2023, "00:00");

#### Move

INSERT INTO Move

VALUES

(1, 1, 1, "2,2"),

(2, 1, 2, "1,1"),

(3, 1, 1, "3,3"),

(4, 1, 2, "3,1"),

(5, 1, 1, "2,1"),

(6, 1, 2, "2,3"),

(7, 1, 1, "3,2"),

(8, 1, 2, "1,2"),

(9, 1, 1, "1,3"),

(1, 2, 1, "4,4"),

(2, 2, 2, "1,1"),

(3, 2, 1, "5,2"),

(4, 2, 2, "5,5"),

(1, 3, 1, "1,1"),

(2, 3, 2, "3,3"),

(3, 3, 1, "5,6"),

(4, 3, 2, "4,3"),

(5, 3, 1, "4,5"),

(6, 3, 2, "5,3"),

(7, 3, 1, "3,4"),

(8, 3, 2, "6,3"),

(9, 3, 1, "2,3");

#### Settings

INSERT INTO Settings (IsSinglePlayer, Topology, GridSize, LineSize)

VALUES

(1, "Plane", 3, 3),

(0, "Torus", 5, 2),

(1, "Projective Plane", 6, 5);

## Test Log

The first part of the test number corresponds to the specific objective being tested for the first five sections. Later tests are white-box to test functionality of the program required to meet the objectives. The tests were not completed (and so the program code was not modified) in the order of the headings as the dependencies of some functions and classes had to be tested first. Also, only a representative sample of tests are included. More tests and modifications to the program were conducted so screenshots in this log may differ from the final version

### 1. General

No tests were conducted as these objectives are more descriptive so are assessed in the evaluation.

### 2. Menu

These tests were conducted sixth.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 2.1.1.1 | N/A | N/A | There is a button, which directs the user to the settings screen | Fail |
| **Actual Outcome:**    The settings button is in the incorrect position and it crashes the program when it is pressed  **Error:** The settings button vertical offset is too small. Also, when buttons are pressed, the program tries to render them differently by reducing each colour component by a value. However, the colours are initialised as strings instead of tuples of integers, which raises an error  **Fix:** I removed the colour parameter from the initialisation method of the button class as all buttons in the program are grey. I also changed the colour attribute to a tuple. Hence, I had to modify every button object in the program. Furthermore, I increased the settings button vertical offset proportion. I also modified other variables to make the settings button look nicer | | | | |
| 2.1.1.2 | N/A | N/A | There is a button, which directs the user to the settings screen | Pass |
| **Actual Outcome:**    The settings button is displayed correctly    When the button is pressed the program correctly directs the user to the settings screen (*nota bene* the blatant errors in the settings screen will be addressed in another test) | | | | |
| 2.1.2.1 | N/A | N/A | There is a button, which directs the user to the load previous game screen | Fail |
| **Actual Outcome:**    No load previous game button is visible to the user  **Error:** When assigning the y attribute of the game button, the settings button y attribute is being treated as a proportion when it is actually a value. Also, the settings button height attribute was not being considered. This was the same issue with the previous game button  **Fix:** I added on the y and height attributes of the settings button to the y attribute of the previous game button | | | | |
| 2.1.2.2 | N/A | N/A | There is a button, which directs the user to the load previous game screen | Fail |
| **Actual Outcome:**    A previous game button is displayed, but its design is incorrect    When the button is pressed the program correctly directs the user to the previous game screen (*nota bene* the blatant errors in the previous game screen will be addressed in another test)  **Error:** The vertical offset of the settings button is too small. Also, the width of the buttons is too small. Furthermore, the button text size is too big  **Fix:** I modified class variables in the Menu class to do with the styling of the buttons. I also modified the font class variables in the Button class | | | | |
| 2.1.2.3 | N/A | N/A | There is a previous game button | Pass |
| **Actual Outcome:**    A previous game button is displayed correctly | | | | |

### 3. Settings Screen

These tests were conducted seventh.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 3.1.1.1 | N/A | N/A | Program displays correct list of topologies | Fail |
| **Actual Outcome:**    The list of topologies is incorrectly displayed  **Error:** The topology prompt is not being displayed, the buttons are too small and the text is too big  **Fix:** I made the render method render the topology. I also changed the functionality of the button class to have a small text option. All of the buttons (bar the return) in the the settings screen and the list of buttons class now use this as they are too big meaning the game mode setting is not displaying. Moreover, I modified the width and height of various buttons so that they would be aesthetically pleasing and allow the game mode list of buttons to be on the screen | | | | |
| 3.1.1.2 | N/A | N/A | Program displays correct list of topologies | Fail |
| **Actual Outcome:**    The list of topologies (including ones in the additional objectives) is correctly displayed but the Plane button is not selected  **Error:** There is a logic error where if the index of the initial selected button is 0 the initialisation method does not select this button as the boolean equivalent of 0 is FALSE  **Fix:** I changed the logic in the selection structure to accommodate this | | | | |
| 3.1.1.3 | N/A | N/A | Program displays correct list of topologies | Pass |
| **Actual Outcome:**    The list of topologies is correctly displayed | | | | |
| 3.1.1.4 | N/A | N/A | User is able to select the Mobius strip topology | Fail |
| **Actual Outcome:**    The Mobius Strip button does highlight when the user presses down on it    However, it does not select when the user releases the button  **Error:** When the list of buttons object is updated, the state of each button is not checked and the select method not called. Also, the same logic error (in 3.1.1.2) appears in the select method when setting the selected attribute of the previously selected button to FALSE  **Fix:** I modified the update method of the ListOfButtons class to check the state of each button and call the select method when required. I also changed the logic in the select method | | | | |
| 3.1.1.5 | N/A | N/A | User is able to select the Mobius strip topology | Pass |
| **Actual Outcome:**    The Mobius Strip button is selected when the user clicks it | | | | |
| 3.1.3.1 | 5 | Boundary | User can change moves input to 5 | Fail |
| **Actual Outcome:**    After clicking the moves character input box and pressing the 5 key, nothing happens  **Error:** The check\_selected method does not return a boolean value but changes the value of an attribute. Also, when a key is pressed, its character code is retrieved in hexadecimal  **Fix:** I modified the check\_selected method so it returns a value and removed the selected attribute. This is because this attribute is unnecessary as the main program loop recognises that the character input box has been deselected once a key has been pressed. In the main program loop, I introduced a selection structure to assign the character if a number key is pressed. If a non-number key is pressed, the program ignores this | | | | |
| 3.1.3.2 | 5 | Boundary | User can change moves input to 5 | Pass |
| **Actual Outcome:**    The moves character input box correctly changes to 5 after the user clicks it and presses the 5 key | | | | |
| 3.1.3.3 | 6 | Erroneous | User can change moves input to 6 and error displays | Fail |
| **Actual Outcome:**    The moves input has changed to 6 but the error message is rendered incorrectly  **Error:** When there is a newline in the error message, pygame cannot render this as multiple lines. Also, due to indentation in the code multiple spaces are being rendered  **Fix:** I modified the render\_error method so that it renders each line of an error separately.  I also changed the error messages so that they don’t contain lots of additional spaces and altered the error message font | | | | |
| 3.1.3.4 | 6 | Erroneous | User can change moves input to 6 and error displays | Pass |
| **Actual Outcome:**    The error has correctly displayed after the user pressed the moves input and the 6 key | | | | |
| 3.1.3.5 | N/A | N/A | Error stops displaying when user changes moves input from 6 to 3 | Fail |
| **Actual Outcome:**    The error message does not stop displaying when the user corrects the moves input back to 3  **Error:** If there is no error, then the error\_msg attribute is not reset to nothing  **Fix:** I added an else statement (in the validate\_inputs method) to reset the error\_msg attribute if all of the error tests failed | | | | |
| 3.1.3.6 | N/A | N/A | Error stops displaying when user changes moves input from 6 to 3 | Pass |
| **Actual Outcome:**      The error message does stop displaying when the user corrects the moves input from 6 to 3 | | | | |
| 3.1.6.1 | N/A | Normal | User returns to menu when they press return button | Pass |
| **Actual Outcome:**      The user is correctly returned to the menu | | | | |
| 3.1.6.2  3.3.1 | N/A | Erroneous | User cannot return to menu when moves input is 0 | Pass |
| **Actual Outcome:**      Pressing and releasing the return button achieves nothing when the moves input is 0 | | | | |

### 4. Load Previous Game Screen

These tests were conducted ninth (last).

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 4.2.1.1  4.2.2.1  4.2.3.1  4.2.4.1 | N/A | N/A | 3 previous games display correctly | Fail |
| **Actual Outcome:**    The load previous game screen is incorrectly displayed  **Error:** Various aspects were wrong and variables not optimised to be aesthetically pleasing. There is also a representational error in Python as it computes 0.333 \* 100 to be 33.300000000000004  **Fix:** To correct the representational error, I used Python’s built-in round function (in the render method) instead of the round\_sf function in utils. I also optimised various variables in the PreviousGame class for aesthetic purposes | | | | |
| 4.2.1.1  4.2.2.1  4.2.3.1  4.2.4.1 | N/A | N/A | 3 previous games display correctly | Pass |
| **Actual Outcome:**    The previous game screen correctly displays:   * Which player won or if the game was a draw * Whether the game was single or multiplayer * The topology of the grid in the game * The date and time the game finished | | | | |
| 4.5.2.1 | N/A | N/A | Win rate of each player displays correctly as 33.3% | Pass |
| **Actual Outcome:**    The win rate of each player correctly displays as 33.3% | | | | |

### 5. Game Screen

These tests were conducted eighth.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 5.2.1 | N/A | N/A | Line displayed correctly | Pass |
| **Purpose:** Test objective 5.2 for a 3x3 plane grid with a line length of 3 needed to win in single-player mode when the Computer wins  **Actual Outcome:**            The screenshots show that a line is correctly displayed | | | | |
| 5.2.2 | N/A | N/A | Line displayed correctly | Pass |
| **Purpose:** Test objective 5.2 for a 4x4 Mobius strip grid with a line length of 3 needed to win in single-player mode when Player 1 wins  **Actual Outcome:**          The line is displayed correctly | | | | |
| 5.2.3 | N/A | N/A | Line displayed correctly | Pass |
| **Purpose:** Test objective 5.2 for a 6x6 Projective plane grid with a line length of 4 needed to win in multiplayer mode when Player 2 wins  **Actual Outcome:**            The line is displayed correctly | | | | |
| 5.2.4 | N/A | N/A | Line does not display | Pass |
| **Purpose:** Test objective 5.2 for a 3x3 Plane grid with a line length of 3 needed to win in multiplayer mode when the game is a draw  **Actual Outcome:**              The line does not display | | | | |
| 5.4.1 | N/A | N/A | Move is undone | Pass |
| **Purpose:** Test objective 5.4 for a multiplayer game mode  **Actual Outcome:**        The move is correctly undone | | | | |
| 5.6.1.1  5.6.2.1  5.6.3.1  5.6.4.1  5.6.5.1 | N/A | N/A | The correct data is present in the database | Pass |
| **Actual Outcome:**                All 3 relations are populated with the correct data. A new Settings record is created as no record for that combination of settings previously existed. The settings record also contains the grid size and line size in accordance with additional objectives 3.1.1 and 3.1.2 | | | | |
| 5.7.2.1 | N/A | N/A | The user is moved forward through the moves | Pass |
| **Actual Outcome:**              Pressing the next button does correctly cycle the user through the moves | | | | |
| 5.7.2.2 | N/A | N/A | The user is moved backward through the moves | Pass |
| **Actual Outcome:**  *Nota bene,* the state of the program persists on from the last test (5.7.2.1)            The user is correctly moved backwards through the moves | | | | |

### 6. Computer Algorithm

These tests were conducted fifth.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 6.1.1 | test.py script | Normal | [] | Fail |
| **Purpose:** Test the count\_outcomes method of the Computer class when no moves lead to winning  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 3, 3, False)  computer = Computer(grid, 4)  print(computer.count\_outcomes(4, []))  **Actual Outcome:**    **Error:** When the recursive function (say where n = m) was returned from, the algorithm was not reversing the move just done in the function, where n = m + 1, allowing for a player to do multiple moves at once  **Fix:** I reprogrammed the algorithm to undo the move after its recursive calling has been returned from | | | | |
| 6.1.2 | test.py script | Normal | [] | Pass |
| **Purpose:** Test the new count\_outcomes method of the Computer class when no moves lead to winning  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 3, 3, False)  computer = Computer(grid, 4)  print(computer.count\_outcomes(4, 1, []))  **Actual Outcome:** | | | | |
| 6.1.3 | test.py script | Normal | 30 | Fail |
| **Purpose:** Test the new count\_outcomes method of the Computer class  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 3, 3, False)  grid.move([2, 2], 1)  grid.move([3, 3], 2)  computer = Computer(grid, 3)  print(computer.count\_outcomes(3, 1, []).count(1))  **Actual Outcome:**    **Error:** There was no error in the count\_outcomes method of the Computer class, rather in the check\_line method of the grid class. The line list was not reset to solely containing the starting square when the direction was changed. Hence, the line list could contain additional squares stemming from the starting square in other directions  **Fix:** I made the method reset the line list to just the starting square when the direction is changed and the line is not long enough | | | | |
| 6.1.4 | test.py script | Normal | 30 | Pass |
| **Purpose:** Test the new count\_outcomes method of the Computer class with the new check\_line method of the Grid class  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 3, 3, False)  grid.move([2, 2], 1)  grid.move([3, 3], 2)  computer = Computer(grid, 3)  print(computer.count\_outcomes(3, 1, []).count(1))  **Actual Outcome:** | | | | |
| 6.1.5 | test.py script | Normal | 20 8 | Pass |
| **Purpose:** Test the new count\_outcomes method of the Computer class with the new check\_line method of the Grid class  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([-1, -1], 3, 3, True)  grid.move([1, 1], 1)  grid.move([2, 2], 2)  grid.move([3, 3], 1)  computer = Computer(grid, 3)  outcomes = computer.count\_outcomes(3, 2, [])  print(outcomes.count(1), outcomes.count(2))  **Actual Outcome:** | | | | |
| 6.2.1 | test.py script | Normal | [1, 2] | Pass |
| **Purpose:** Test the get\_best\_move method of the Computer class where no moves lead to winning or losing  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 4, 3, False)  grid.move([1, 1], 1)  computer = Computer(grid, 3)  print(computer.get\_best\_move())  **Actual Outcome:** | | | | |
| 6.2.2 | test.py script | Normal | [3, 3] | Fail |
| **Purpose:** Test the get\_best\_move method of the Computer class  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 3, 3, False)  grid.move([2, 2], 1)  grid.move([2, 3], 2)  grid.move([1, 2], 1)  grid.move([3, 2], 2)  grid.move([1, 1], 1)  computer = Computer(grid, 3)  print(computer.get\_best\_move())  **Actual Outcome:**    **Error:** There are two errors causing this failure. Firstly, there is a typo meaning that the loop does not iterate through the final column. Secondly, when the count\_outcomes method is called for the first time, the recursion depth (n) is not reduced by 1 so the algorithm examines one more move than required  **Fix:** I amended the typo. I also subtracted one from the recursion depth when calling the count\_outcomes subroutine | | | | |
| 6.2.3 | test.py script | Normal | [3, 3] | Pass |
| **Purpose:** Test the new get\_best\_move method of the Computer class with the new count\_outcomes method  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([0, 0], 3, 3, False)  grid.move([2, 2], 1)  grid.move([2, 3], 2)  grid.move([1, 2], 1)  grid.move([3, 2], 2)  grid.move([1, 1], 1)  computer = Computer(grid, 3)  print(computer.get\_best\_move())  **Actual Outcome:** | | | | |
| 6.2.4 | test.py script | Normal | [2, 3] | Pass |
| **Purpose:** Test the new get\_best\_move method of the Computer class with the new count\_outcomes method  **test.py:**  from MOD.computer import Computer  from MOD.grid import Grid  grid = Grid([1, -1], 4, 3, True)  grid.move([1, 1], 1)  grid.move([4, 3], 2)  grid.move([3, 4], 1)  computer = Computer(grid, 3)  print(computer.get\_best\_move())  **Actual Outcome:** | | | | |

### 7. Grid

These tests were conducted third.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 7.1.1 | test.py script | Normal | Program displays grid correctly | Pass |
| **Purpose:** Test the render and \_\_init\_\_ methods of the Grid class  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((0, 0), 3, 3, False)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  pygame.display.update()  **Actual Outcome:**    The grid is displayed correctly with:   * No arrows for a plane topology * A grid size of 3x3 * No adjacency map displayed | | | | |
| 7.1.2 | test.py script | Normal | Program displays grid correctly | Pass |
| **Purpose:** Test the render and \_\_init\_\_ methods of the Grid class  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((1, 1), 5, 3, True)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  pygame.display.update()  **Actual Outcome:**    The grid is displayed correctly with:   * Upward and left arrows for a torus topology * A grid size of 5x5 * An adjacency map displayed | | | | |
| 7.1.3 | test.py script | Normal | Program displays grid correctly | Pass |
| **Purpose:** Test the render and \_\_init\_\_ methods of the Grid class  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((-1, -1), 6, 5, False)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  pygame.display.update()  **Actual Outcome:**    The grid is displayed correctly with:   * Upward/downward and left/right arrows for a projective plane topology * A grid size of 6x6 * No adjacency map displayed | | | | |
| 7.2.1 | test.py script | Normal | Console outputs (3, 2) when the cell 3 across and 4 down is clicked | Fail |
| **Purpose:** Test the get\_state method of the Grid class  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((0, 0), 6, 5, False)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  grid\_state = grid.get\_state(WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  if grid\_state:  print(grid\_state)  WIN.fill("white")  grid.render(WIN)  pygame.display.update()  **Actual Outcome:**    **Error:** There is a typo on line 39 where the method compares the upper and lower limits of the row to the mouse’s x position  **Fix:** I amended the typo so the method compares the limits to the mouse’s y position | | | | |
| 7.2.2 | test.py script | Normal | Console outputs (3, 2) when the cell 3 across and 4 down is clicked | Pass |
| **Purpose:** Test the new get\_state method of the Grid class  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((0, 0), 6, 5, False)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  grid\_state = grid.get\_state(WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  if grid\_state:  print(grid\_state)  WIN.fill("white")  grid.render(WIN)  pygame.display.update()  **Actual Outcome:** | | | | |
| 7.2.3 | test.py script | Erroneous | No console output when outside grid is clicked | Pass |
| **Purpose:** Test the get\_state method of the Grid class  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((0, 0), 6, 5, False)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  grid\_state = grid.get\_state(WIN, mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  if grid\_state:  print(grid\_state)  WIN.fill("white")  grid.render(WIN)  pygame.display.update()  **Actual Outcome:** | | | | |
| 7.3.1 | test.py script | Erroneous | [1, 1] [1, 2] [1, 3]  [2, 1] [2, 2] [2, 3]  [3, 1] [3, 2] [3, 3] | Pass |
| **Purpose:** Test the corr\_square method of the Grid class when parsed a real square on the grid  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 0), 3, 3, False)  for row in range(1, 4):  for col in range(1, 4):  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.3.2 | test.py script | Erroneous | [0, 0] [0, 1] [0, 2] [0, 3] [0, 4]  [1, 0] [1, 4]  [2, 0] [2, 4]  [3, 0] [3, 4]  [4, 0] [4, 1] [4, 2] [4, 3] [4, 4] | Pass |
| **Purpose:** Test the corr\_square method of the Grid class for a plane topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 0), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.3.3 | test.py script | Normal & Erroneous | [0, 3] [0, 1] [0, 2] [0, 3] [0, 1]  [1, 3] [1, 1]  [2, 3] [2, 1]  [3, 3] [3, 1]  [4, 3] [4, 1] [4, 2] [4, 3] [4, 1] | Pass |
| **Purpose:** Test the corr\_square method of the Grid class for a cylinder topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 1), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.3.4 | test.py script | Normal & Erroneous | [4, 3] [0, 1] [0, 2] [0, 3] [4, 1]  [3, 3] [3, 1]  [2, 3] [2, 1]  [1, 3] [1, 1]  [0, 3] [4, 1] [4, 2] [4, 3] [0, 1] | Fail |
| **Purpose:** Test the corr\_square method of the Grid class for a Mobius strip topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((0, -1), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:**    **Error:** If the left and right sides are twisted (so topology[1] is -1), the function reflects the column about the middle instead of the row  **Fix:** I corrected the function so it reflects the row about the middle if the left and right sides are twisted. Consequently, it will reflect the column about the middle if the top and bottom and sides are twisted | | | | |
| 7.3.5 | test.py script | Normal & Erroneous | [4, 3] [0, 1] [0, 2] [0, 3] [4, 1]  [3, 3] [3, 1]  [2, 3] [2, 1]  [1, 3] [1, 1]  [0, 3] [4, 1] [4, 2] [4, 3] [0, 1] | Pass |
| **Purpose:** Test the new corr\_square method of the Grid class for a Mobius strip topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((0, -1), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.3.6 | test.py script | Normal | [3, 3] [3, 1] [3, 2] [3, 3] [3, 1]  [1, 3] [1, 1]  [2, 3] [2, 1]  [3, 3] [3, 1]  [1, 3] [1, 1] [1, 2] [1, 3] [1, 1] | Pass |
| **Purpose:** Test the new corr\_square method of the Grid class for a torus topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((1, 1), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.3.7 | test.py script | Normal | [1, 3] [3, 1] [3, 2] [3, 3] [1, 1]  [3, 3] [3, 1]  [2, 3] [2, 1]  [1, 3] [1, 1]  [3, 3] [1, 1] [1, 2] [1, 3] [3, 1] | Pass |
| **Purpose:** Test the new corr\_square method of the Grid class for a Klein bottle topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((1, -1), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.3.8 | test.py script | Normal | [1, 1] [3, 3] [3, 2] [3, 1] [1, 3]  [3, 3] [3, 1]  [2, 3] [2, 1]  [1, 3] [1, 1]  [3, 1] [1, 3] [1, 2] [1, 1] [3, 3] | Pass |
| **Purpose:** Test the new corr\_square method of the Grid class for a projective plane topology when parsed an imaginary square on the adjacency map  **test.py:**  from MOD.grid import Grid  grid = Grid((-1, -1), 3, 3, False)  for row in range(0, 5):  for col in range(0, 5):  if 1 <= row <= 3 and 1 <= col <= 3:  continue  print(grid.corr\_square([row, col]), end=" ")  print()  **Actual Outcome:** | | | | |
| 7.4.1 | test.py script | Normal | 0 0 0 0 1  0 1 0 0 0  2 0 0 2 0  0 0 0 0 1  0 1 0 0 0 | Pass |
| **Purpose:** Test the update\_grid method of the Grid class  **test.py:**  from MOD.grid import Grid  grid = Grid((1, -1), 3, 3, True)  grid.update\_grid([1, 1], 1)  grid.update\_grid([2, 3], 2)  grid.update\_grid([3, 2], 2)  grid.update\_grid([3, 2], 0)  for row in grid.grid:  [print(cell, end=" ") for cell in row]  print()  **Actual Outcome:** | | | | |
| 7.5.1 | test.py script | Normal | [[3, 3], [2, 2], [1, 1]] | Fail |
| **Purpose:** Test the check\_line method of the Grid class on a plane topology 3x3 grid  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 0), 3, 3, True)  grid.update\_grid([1, 1], 1)  grid.update\_grid([1, 2], 2)  grid.update\_grid([3, 2], 1)  grid.update\_grid([3, 1], 2)  grid.update\_grid([2, 2], 1)  grid.update\_grid([2, 3], 2)  grid.update\_grid([3, 3], 1)  print(grid.check\_line([3, 3], 1))  **Actual Outcome:**    **Error:** The square variable was not being updated each time the algorithm moved on to a different square and the starting square was not added to the line list  **Fix:** I parsed [row, col] to the adjacent square method to update the square and made the initial value of the line list a list whose only element is the starting square | | | | |
| 7.5.2 | test.py script | Normal | [[3, 3], [2, 2], [1, 1]] | Pass |
| **Purpose:** Test the new check\_line method of the Grid class on a plane topology 3x3 grid  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 0), 3, 3, True)  grid.update\_grid([1, 1], 1)  grid.update\_grid([1, 2], 2)  grid.update\_grid([3, 2], 1)  grid.update\_grid([3, 1], 2)  grid.update\_grid([2, 2], 1)  grid.update\_grid([2, 3], 2)  grid.update\_grid([3, 3], 1)  print(grid.check\_line([3, 3], 1))  **Actual Outcome:** | | | | |
| 7.5.3 | test.py script | Erroneous | False | Pass |
| **Purpose:** Test the new check\_line method of the Grid class on a cylinder topology 4x4 grid  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 1), 4, 4, False)  grid.update\_grid([2, 2], 1)  grid.update\_grid([4, 4], 2)  grid.update\_grid([3, 4], 1)  grid.update\_grid([1, 2], 2)  grid.update\_grid([4, 3], 1)  grid.update\_grid([2, 3], 2)  grid.update\_grid([2, 1], 1)  print(grid.check\_line([2, 1], 1))  **Actual Outcome:** | | | | |
| 7.5.4 | test.py script | Normal | [[1, 1], [5, 2], [4, 3], [3, 4]] | Fail |
| **Purpose:** Test the new check\_line method of the Grid class on a torus topology 5x5 grid with a 4-square line needed to win  **test.py:**  from MOD.grid import Grid  grid = Grid((1, 1), 5, 4, False)  grid.update\_grid([3, 3], 1)  grid.update\_grid([5, 2], 2)  grid.update\_grid([2, 2], 1)  grid.update\_grid([4, 3], 2)  grid.update\_grid([4, 4], 1)  grid.update\_grid([3, 4], 2)  grid.update\_grid([4, 5], 1)  grid.update\_grid([1, 1], 2)  print(grid.check\_line([1, 1], 2))  **Actual Outcome:**    **Error:** The row and col variables were not reassigned to the starting square each time a particular direction failed to give a winning line  **Fix:** I changed to instantiation of the row and col variables so they are reassigned each time a particular directions fails to give a winning line | | | | |
| 7.5.5 | test.py script | Normal | [[1, 1], [5, 2], [4, 3], [3, 4]] | Pass |
| **Purpose:** Test the new check\_line method of the Grid class on a torus topology 5x5 grid with a 4-square line needed to win  **test.py:**  from MOD.grid import Grid  grid = Grid((1, 1), 5, 4, False)  grid.update\_grid([3, 3], 1)  grid.update\_grid([5, 2], 2)  grid.update\_grid([2, 2], 1)  grid.update\_grid([4, 3], 2)  grid.update\_grid([4, 4], 1)  grid.update\_grid([3, 4], 2)  grid.update\_grid([4, 5], 1)  grid.update\_grid([1, 1], 2)  print(grid.check\_line([1, 1], 2))  **Actual Outcome:** | | | | |
| 7.5.6 | test.py script | Normal | [[6, 3], [5, 2], [4, 1],  [1, 3], [2, 2], [3, 1]] | Fail |
| **Purpose:** Test the new check\_line method of the Grid class on a projective plane topology 6x6 grid  **test.py:**  from MOD.grid import Grid  grid = Grid((-1, -1), 6, 6, False)  grid.update\_grid([3, 2], 1)  grid.update\_grid([3, 1], 2)  grid.update\_grid([4, 3], 1)  grid.update\_grid([2, 2], 2)  grid.update\_grid([6, 1], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([2, 4], 1)  grid.update\_grid([4, 1], 2)  grid.update\_grid([2, 6], 1)  grid.update\_grid([5, 2], 2)  grid.update\_grid([6, 6], 1)  grid.update\_grid([6, 3], 2)  print(grid.check\_line([6, 3], 2))  **Actual Outcome:**    **Error:** Firstly, the algorithm does not reverse the direction of the diagonal line when it moves between twisted edges. Secondly, it does not check both directions when the parsed square is in the middle of a line  **Fix:** I reversed the direction of the diagonal line when the algorithm moves between twisted edges and ensured that the algorithm checks both directions for a line | | | | |
| 7.5.7 | test.py script | Normal | [[6, 3], [5, 2], [4, 1], [1, 3], [2, 2], [3, 1]] | Pass |
| **Purpose:** Test the new check\_line method of the Grid class on a projective plane topology 6x6 grid  **test.py:**  from MOD.grid import Grid  grid = Grid((-1, -1), 6, 6, False)  grid.update\_grid([3, 2], 1)  grid.update\_grid([3, 1], 2)  grid.update\_grid([4, 3], 1)  grid.update\_grid([2, 2], 2)  grid.update\_grid([6, 1], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([2, 4], 1)  grid.update\_grid([4, 1], 2)  grid.update\_grid([2, 6], 1)  grid.update\_grid([5, 2], 2)  grid.update\_grid([6, 6], 1)  grid.update\_grid([6, 3], 2)  print(grid.check\_line([6, 3], 2))  **Actual Outcome:** | | | | |
| 7.6.1 | test.py script | Normal | [1, 0]: h  [1, 1]: h  [1, 2]: h | Pass |
| **Purpose:** Test the get\_orientations method of the Grid class  **test.py:**  from MOD.grid import Grid  grid = Grid((0, 0), 3, 3, False)  orientations = grid.get\_orientations([[1, 0], [1, 1], [1, 2]])  for square in orientations:  print(f"{square}: {orientations[square]}")  **Actual Outcome:** | | | | |
| 7.6.2 | test.py script | Normal | [3, 1]: db  [2, 4]: db  [1, 3]: db  [4, 2]: db | Pass |
| **Purpose:** Test the get\_orientations method of the Grid class  **test.py:**  from MOD.grid import Grid  grid = Grid((1, 1), 4, 4, True)  orientations = grid.get\_orientations([[3, 1], [2, 4], [1, 3], [4, 2]])  for square in orientations:  print(f"{square}: {orientations[square]}")  **Actual Outcome:** | | | | |
| 7.6.3 | test.py script | Normal | [2, 2]: v  [1, 2]: v  [5, 4]: v  [4, 4]: v | Pass |
| **Purpose:** Test the get\_orientations method of the Grid class  **test.py:**  from MOD.grid import Grid  grid = Grid((-1, -1), 5, 4, False)  orientations = grid.get\_orientations([[2, 2], [1, 2], [5, 4], [4, 4]])  for square in orientations:  print(f"{square}: {orientations[square]}")  **Actual Outcome:** | | | | |
| 7.6.4 | test.py script | Normal | [6, 3]: db  [5, 2]: db  [4, 1]: db  [1, 3]: df  [2, 2]: df  [3, 1]: df | Fail |
| **Purpose:** Test the get\_orientations method of the Grid class  **test.py:**  from MOD.grid import Grid  grid = Grid((-1, -1), 6, 6, True)  orientations = grid.get\_orientations([[6, 3], [5, 2], [4, 1], [1, 3], [2, 2], [3, 1]])  for square in orientations:  print(f"{square}: {orientations[square]}")  **Actual Outcome:**    **Error:** The first square in the line list was in the middle of the line (on the grid), which raised a number of issues. A direction was not assigned to square [4, 1] as there were other squares in the line list and [4, 1] was one end of a line. Also, when going the other direction for square [6, 3] the next square was on the previously explored upper half of the line, which meant that the algorithm did not assign directions to the other squares  **Fix:** If the square was after the starting square, I assigned a direction to it regardless of whether there was a square after it on the line. If the square was the starting square, I assigned a direction to it only after the algorithm has moved, in that direction, to another square on the line. Also, I expanded the check for the next square being on the line to evaluate to true if the next square is anywhere in the line (not just after the current square) | | | | |
| 7.6.5 | test.py script | Normal | [6, 3]: db  [5, 2]: db  [4, 1]: db  [1, 3]: df  [2, 2]: df  [3, 1]: df | Pass |
| **Purpose:** Test the new get\_orientations method of the Grid class  **test.py:**  from MOD.grid import Grid  grid = Grid((-1, -1), 6, 6, True)  orientations = grid.get\_orientations([[6, 3], [5, 2], [4, 1], [1, 3], [2, 2], [3, 1]])  for square in orientations:  print(f"{square}: {orientations[square]}")  **Actual Outcome:** | | | | |
| 7.7.1 | test.py script | Normal | Program displays grid & line correctly | Fail |
| **Purpose:** Test the render\_line and render methods of the Grid class for a 3x3 plane topology  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((0, 0), 3, 3, False)  grid.update\_grid([2, 1], 1)  grid.update\_grid([1, 1], 2)  grid.update\_grid([2, 2], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([2, 3], 1)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  grid.render\_line(WIN, [[2, 1], [2, 2], [2, 3]])  pygame.display.update()  **Actual Outcome:**    The grid is displayed incorrectly with the noughts filled-in instead of just circumferences. Also, the line is displayed in the incorrect position  **Error:** The render\_linemethod has drawn the line one square too far down and right as it assumed that array indexing for the grid starts as 0. It also rendered filled-in circles instead of their outlines  **Fix:** I changed the render\_line method so it utilises the fact that array indexing for the grid starts at 1. I also amended the arguments parsed into the circle drawing procedure so that it only draws the circumference | | | | |
| 7.7.2 | test.py script | Normal | Program displays grid & line correctly | Pass |
| **Purpose:** Test the new render\_line and render methods of the Grid class for 3x3 plane topology  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((0, 0), 3, 3, False)  grid.update\_grid([2, 1], 1)  grid.update\_grid([1, 1], 2)  grid.update\_grid([2, 2], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([2, 3], 1)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  grid.render\_line(WIN, [[2, 1], [2, 2], [2, 3]])  pygame.display.update()  **Actual Outcome:**    The grid is displayed correctly with both the noughts and crosses correct. The line is displayed correctly and in the correct place | | | | |
| 7.7.3 | test.py script | Normal | Program displays grid & line correctly | Pass |
| **Purpose:** Test the new render\_line and render methods of the Grid class for a 4x4 torus topology  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((1, 1), 4, 4, True)  grid.update\_grid([1, 1], 1)  grid.update\_grid([3, 1], 2)  grid.update\_grid([1, 2], 1)  grid.update\_grid([2, 4], 2)  grid.update\_grid([1, 4], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([3, 4], 1)  grid.update\_grid([4, 2], 2)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  grid.render\_line(WIN, [[3, 1], [2, 4], [1, 3], [4, 2]])  pygame.display.update()  **Actual Outcome:**    The grid and line are displayed correctly with:   * The noughts and crosses on the grid correct * The noughts and crosses in the adjacency map correct * The line correct | | | | |
| 7.7.4 | test.py script | Normal | Program displays grid & line correctly | Pass |
| **Purpose:** Test the new render\_line and render methods of the Grid class for a 5x5 projective plane topology  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((-1, -1), 5, 4, False)  grid.update\_grid([2, 2], 1)  grid.update\_grid([3, 1], 2)  grid.update\_grid([1, 2], 1)  grid.update\_grid([2, 4], 2)  grid.update\_grid([5, 4], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([4, 4], 1)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  grid.render\_line(WIN, [[2, 2], [1, 2], [5, 4], [4, 4]])  pygame.display.update()  **Actual Outcome:**    The grid, noughts and crosses and line are all displayed correctly | | | | |
| 7.7.5 | test.py script | Normal | Program displays grid & line correctly | Pass |
| **Purpose:** Test the new render\_line and render methods of the Grid class for a 6x6 projective plane topology  **test.py:**  import pygame  from MOD.grid import Grid  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  grid = Grid((-1, -1), 6, 6, True)  grid.update\_grid([1, 6], 1)  grid.update\_grid([6, 3], 2)  grid.update\_grid([1, 2], 1)  grid.update\_grid([5, 2], 2)  grid.update\_grid([5, 4], 1)  grid.update\_grid([4, 1], 2)  grid.update\_grid([4, 4], 1)  grid.update\_grid([1, 3], 2)  grid.update\_grid([4, 3], 1)  grid.update\_grid([2, 2], 2)  grid.update\_grid([4, 2], 1)  grid.update\_grid([3, 1], 2)  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  grid.render(WIN)  grid.render\_line(WIN, [[6, 3], [5, 2], [4, 1], [1, 3], [2, 2], [3, 1]])  pygame.display.update()  **Actual Outcome:** | | | | |

### 8. Inputs

These tests were conducted second.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 8.1.1 | test.py script | Normal | Program displays a red button with the word “Test” | Pass |
| **Purpose:** Test the render and \_\_init\_\_ methods of the Button class  **test.py:**  import pygame  from MOD.inputs import Button  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  button = Button(300, 250, 200, 100, "Test", "red")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  button.render(WIN)  pygame.display.update()  **Actual Outcome:**    The button is displayed correctly | | | | |
| 8.1.2 | test.py script | Normal | Program displays a button, which darkens when clicked | Fail |
| **Purpose:** Test the update method of the Button class  **test.py:**  import pygame  from MOD.inputs import Button  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  button = Button(300, 250, 200, 100, "Test", "white")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  button.update(mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  button.render(WIN)  pygame.display.update()  **Actual Outcome:**      The colour of the unpressed button in the first image is indifferent to the colour of the pressed button in the second image  **Error:** Else statements were not being used meaning that the button state was being changed when it should not have been  **Fix:** I added the else statements where necessary so that the button state was not updated to 0 regardless of a condition being met | | | | |
| 8.1.3 | test.py script | Normal | Program displays a button, which darkens when clicked | Pass |
| **Purpose:** Test the new update method of the Button class  **test.py:**  import pygame  from MOD.inputs import Button  pygame.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  button = Button(300, 250, 200, 100, "Test", "white")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  button.update(mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  button.render(WIN)  pygame.display.update()  **Actual Outcome:**      The colour of the unpressed button in the first image is darker than the colour of the pressed button in the second image | | | | |
| 8.2.1 | test.py script | Normal | Program displays char input box | Pass |
| **Purpose:** Test the render method of the CharInputBox class  **test.py:**  import pygame  from MOD.inputs import CharInputBox  pygame.init()  pygame.font.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  prompt\_font = pygame.font.SysFont("arial", 30)  prompt = prompt\_font.render("Test", True, "red")  char\_input\_box = CharInputBox(400, 300, prompt, "A")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  WIN.fill("white")  char\_input\_box.render(WIN)  pygame.display.update()  **Actual Outcome:**    The character input box is rendered correctly | | | | |
| 8.2.2 | test.py script | Normal | Console outputs “Selected” when char input box is clicked | Pass |
| **Purpose:** Test the check\_selected method of the CharInputBox class  **test.py:**  import pygame  from MOD.inputs import CharInputBox  pygame.init()  pygame.font.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  prompt\_font = pygame.font.SysFont("arial", 30)  prompt = prompt\_font.render("Test", True, "red")  char\_input\_box = CharInputBox(400, 300, prompt, "A")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  char\_input\_box.check\_selected(mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  if char\_input\_box.selected:  print("Selected")  char\_input\_box.selected = False  WIN.fill("white")  char\_input\_box.render(WIN)  pygame.display.update()  **Actual Outcome:** | | | | |
| 8.2.3 | test.py script | Erroneous | Console does not output “Selected” when screen around char input box is clicked | Pass |
| **Purpose:** Test the check\_selected method of the CharInputBox class  **test.py:**  import pygame  from MOD.inputs import CharInputBox  pygame.init()  pygame.font.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  prompt\_font = pygame.font.SysFont("arial", 30)  prompt = prompt\_font.render("Test", True, "red")  char\_input\_box = CharInputBox(400, 300, prompt, "A")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  mouse\_x, mouse\_y = pygame.mouse.get\_pos()  char\_input\_box.check\_selected(mouse\_x, mouse\_y, pygame.mouse.get\_pressed()[0])  if char\_input\_box.selected:  print("Selected")  char\_input\_box.selected = False  WIN.fill("white")  char\_input\_box.render(WIN)  pygame.display.update()  **Actual Outcome:** | | | | |
| 8.2.4 | test.py script | Normal | Char input box updates when 0 key is pressed | Pass |
| **Purpose:** Test the update\_input method of the CharInputBox class  **test.py:**  import pygame  from MOD.inputs import CharInputBox  pygame.init()  pygame.font.init()  WIN = pygame.display.set\_mode((800, 600))  run = True  prompt\_font = pygame.font.SysFont("arial", 30)  prompt = prompt\_font.render("Test", True, "red")  char\_input\_box = CharInputBox(400, 300, prompt, "A")  while run:  for event in pygame.event.get():  if event.type == pygame.QUIT:  run = False  break  if event.type == pygame.KEYDOWN:  if event.key == pygame.K\_0:  char\_input\_box.update\_input("0")  WIN.fill("white")  char\_input\_box.render(WIN)  pygame.display.update()  **Actual Outcome:**      The char input box is updated to 0 in the second image when the 0 key is pressed | | | | |

### 9. SQL

These tests were conducted fourth. The data used in each test also contains the test database.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 9.1.1 | offset = 0 | Normal | [(3, 1, 1, 'Projective Plane', 6, 1, 1, 2023, '00:00'), (2, 2, 0, 'Torus', 5, 25, 12, 2022, '12:00'), (1, 0, 1, 'Plane', 3, 16, 11, 2022, '20:30')] | Pass |
| **Purpose:** Test the retrieve\_previous\_games function  **Actual Outcome:** | | | | |
| 9.1.2 | offset = 2 | Normal | [(1, 0, 1, 'Plane', 3, 16, 11, 2022, '20:30')] | Pass |
| **Purpose:** Test the retrieve\_previous\_games function  **Actual Outcome:** | | | | |
| 9.1.3 | offset = 3 | Erroneous | [] | Pass |
| **Purpose:** Test the retrieve\_previous\_games function  **Actual Outcome:** | | | | |
| 9.2.1 | N/A | Normal | 3 | Pass |
| **Purpose:** Test the calculate\_total\_games function  **Actual Outcome:** | | | | |
| 9.3.1 | N/A | Normal | (0.3333333333333333, 0.3333333333333333) | Pass |
| **Purpose:** Test the retrieve \_win\_rate function  **Actual Outcome:** | | | | |
| 9.4.1 | settings\_id = 1, outcome = 2, day = 14, month = 3, year = 2053, time = “03:14” | Normal | Record correctly inserted into Game relation | Fail |
| **Purpose:** Test the insert\_game function  **Actual Outcome:**    **Error:** The quotation marks around the time argument were removed when it was inserted (using string interpolation) into the query f-string  **Fix:** I used an escape character to insert quotation marks around the time argument in the query f-string | | | | |
| 9.4.2 | settings\_id = 1, outcome = 2, day = 14, month = 3, year = 2053, time = “03:14” | Normal | Record correctly inserted into Game relation | Pass |
| **Purpose:** Test the new insert\_game function  **Actual Outcome:**      The record has been inserted correctly | | | | |
| 9.5.1 | game\_id = 2 | Normal | (“Torus”, 5, 2, 0) | Fail |
| **Purpose:** Test the retrieve\_settings function  **Actual Outcome:**    **Error:** The specific SettingsID associated with the particular game (identified by the parsed game\_id) was not obtained by the nested query. The SettingsID of the first record in the Game relation was retrieved instead  **Fix:** I added a where clause to the nested query so that GameID matches the parsed one meaning that the SettingsID of the specific game is selected | | | | |
| 9.5.2 | game\_id = 2 | Normal | (“Torus”, 5, 2, 0) | Pass |
| **Purpose:** Test the new retrieve\_settings function  **Actual Outcome:** | | | | |

### 10. Utils

These tests were conducted first.

| **#** | **Data** | **Data Type** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- | --- |
| 10.1.1 | point\_x = 5, point\_y = 7, rect\_x = 3, rect\_y = 4, rect\_w = 10, rect\_h = 6 | Normal | True | Pass |
| **Purpose:** Test the point\_in\_rect function if the point is in the rect  **Actual Outcome** | | | | |
| 10.1.2 | point\_x = 5, point\_y = 7, rect\_x = 5, rect\_y = 4, rect\_w = 2, rect\_h = 3 | Boundary | True | Pass |
| **Purpose:** Test the point\_in\_rect function if the point is on the boundary of the rect  **Actual Outcome:** | | | | |
| 10.1.3 | point\_x = 5, point\_y = 7, rect\_x = 6, rect\_y = 1, rect\_w = 8, rect\_h = 5 | Erroneous | False | Pass |
| **Purpose:** Test the point\_in\_rect function if the point is outside of the rect  **Acutal Outcome** | | | | |
| 10.2.1 | value = 5.135, sf = 3 | Normal | 5.14 | Fail |
| **Purpose:** Test the round\_sf function with normal data  **Acutal Outcome:**    The program did not exit from the function  **Error:** The count was not being incremented if the character in the value was the decimal point  **Fix:** I reprogrammed the function making use of the built-in round function | | | | |
| 10.2.2 | value = 5.146, sf = 3 | Normal | 5.15 | Pass |
| **Purpose:** Test the new round\_sf function with normal data  **Actual Outcome:** | | | | |
| 10.2.3 | value = 5.13329, sf = 4 | Normal | 5.133 | Pass |
| **Purpose:** Test the new round\_sf function with normal data  **Actual Outcome:** | | | | |

# Evaluation

## Objectives Evaluation

### Core Objectives

| **#** | **Objective** | **Met** | **Comment** |
| --- | --- | --- | --- |
| 1 | General | | |
| 1.1 | The program should be written in Python 3.10.5 with Pygame 2.1.0 as the only 3rd party module | Y | I have written the code in Python 3.10.5 and Pygame 2.1.0 is the only 3rd party module |
| 1.2 | The program should have a clean, user-friendly GUI | Y | According to end-user feedback, the program UI is clean and user-friendly |
| 1.3 | The program should be able to run on any machine with Python 3.10.5 and Pygame 2.1.0 installed | Y | I tested it on a computer with Python 3.10.5 and Pygame 2.1.0 installed so I know it runs on them |
| 2 | Menu | | |
| 2.1 | There should be a button to:   * Direct the user to the settings screen * Direct the user to the load previous game screen * Load a game based on the settings and direct the user to the game screen | Y | The settings screen contains three buttons, which achieve all of these functions |
| 3 | Settings Screen | | |
| 3.1 | The user should be able to:   * Select one topology from a list * Enable or disable the ability to undo moves, using a button * Change the number of moves the computer single-player algorithm can foresee ahead (which is actually the recursion depth of the algorithm), using a character input box * Enable or disable the display of an adjacency map, using a button * Change whether the game is single-player or multiplayer * Return to the menu screen, using a button, unless at least one input contains abnormal data | Y | The user is able to do all of these functions |
| 3.2 | The following topologies should be supported:   * Plane * Cylinder * Mobius strip * Torus | Y | The user is able to select all of these topologies from a list of buttons |
| 3.3 | The number of moves the computer single-player algorithm can foresee ahead must be between 1 and 5 inclusive | Y | If the user changes the moves to a number not between 1 and 5, an error message displays and they are not able to return from the settings screen to the menu screen |
| 4 | Load Previous Game Screen | | |
| 4.1 | There should be a list of up to five previous games displaying at any given time | Y | Only 1 to 5 previous games display at any time |
| 4.2 | Each item of the list should have:   * Which player won or if the game was a draw * Whether the game was single or multiplayer * The topology of the grid in the game * The date and time the game finished | Y | All of these data are displayed alongside a previous game |
| 4.3 | When an item is pressed, it should redirect the user to the game screen with that game loaded | Y | Pressing the load button on a previous game loads the game in the game screen |
| 4.4 | There should be a button to:   * Return to the menu * Move to the next list of items (if there are more previous games) * Move to the previous list of items (if the user has moved forward) | Y | All of these buttons are present. The next button only displays and works if there are more earlier previous games. The previous button only displays and works if there are more later previous games |
| 4.5 | The following statistics should be displayed:   * Total number of previous games * Win rate of each player | Y | Each statistic is displayed at the bottom of the screen |
| 5 | Game Screen | | |
| 5.1 | The program should display: | | |
| 5.1.1 | A grid with noughts and crosses on it and arrows to denote the (user set) topology | Y | The user can set the topology of the grid in the settings screen. The grid does display |
| 5.1.2 | An adjacency map (if enabled) | Y | An adjacency map does does display if the user enabled it in the settings screen |
| 5.1.3 | Who’s turn it is if nobody has won | Y | The game screen displays whose turn it is if nobody has won |
| 5.1.4 | An undo button (if enabled and the game is not over) | Y | If the user enabled an undo button in the settings screen and the game is not over, an undo button displays |
| 5.1.5 | A button to exit to the menu (if the game is over) | Y | When the game ends, a return button displays |
| 5.2 | At the end of the game, the program should display the outcome and a line (if a player won) | Y | When the game ends, a line displays (if someone won) and the outcome displays |
| 5.3 | The user should be able to: | | |
| 5.3.1 | Tap a square on the grid to place their nought or cross | Y | Pressing the grid lets the user make their move |
| 5.3.2 | Exit back to the menu at the end of the game | Y | The user can press the return button at the end of the game to exit back to the menu |
| 5.4 | Pressing the undo button should undo the last move in multiplayer mode and last two moves in single-player mode (if enabled) | Y | If the user enables the undo button, pressing it undoes the last move in multiplayer mode and the last two moves in single-player mode |
| 5.5 | The program should | | |
| 5.5.1 | Change player turns after each move | Y | After a user or computer makes their move, the program changes turn |
| 5.5.2 | Automatically use the computer algorithm to make a move (if the game is single player mode) | Y | If the user set the game mode as single player, the computer makes a move when it is its turn |
| 5.5.3 | End the game if appropriate, checking if it is after each move | Y | If a player or the computer has won or the game is a draw, the game will end |
| 5.6 | When a game ends, the program should save to a database:   * The outcome * Each move * The date and time the game finished * The topology of the grid in the game * Whether the game was single or multiplayer | Y | All of these data are stored to a database when a game ends |
| 5.7 | If a previous game has been loaded: | | |
| 5.7.1 | All objectives, with prefix 5, apply with the following exceptions | | |
| 5.7.2 | There should be a forward and a backwards button to allow the user to go to the next or previous move | Y | The user can cycle through the moves in the game using a forwards and backwards button |
| 5.7.3 | There should not be an undo button | Y | An undo button will not display if the user has loaded a previous game |
| 5.7.4 | The user should not be able to tap a square on the grid | Y | The grid does not respond to the user tapping it if the user has loaded a previous game |
| 5.7.5 | All objectives, with prefix 5.5 and 5.6, do not apply | Y | The program does not change player turns, make computer moves or end the game. No data is saved to the database |
| 5.7.6 | Objectives 5.1.5 and 5.3.2 apply throughout the game | Y | The user can always exit to the main menu using a return button |

### Additional Objectives

| **#** | **Objective** | **Met** | **Comment** |
| --- | --- | --- | --- |
| 1 | Settings Screen | | |
| 1.1 | The user should be able to:   * Change the length of the square grid, using a character input box * Change the length of the line needed to win, using a character input box | Y | The user is able to do both of these function in the settings screen |
| 1.2 | The following topologies should be supported:   * Klein bottle * Projective plane | Y | The user can select both of these topologies from a list of buttons in the settings screen |
| 1.3 | The following dimensions should be supported:   * 3x3 * 4x4 * 5x5 * 6x6 | Y | The user can select all of these grid sizes in the settings screen |
| 1.4 | The length of the line needed to win must be greater than zero and less than or equal to the length of the grid | Y | If the length of the line needed to win is not withing those boundaries, an error message displays and the user can return from the settings screen to the menu screen |
| 2 | Load Previous Game Screen | | |
| 2.1 | Each item of the list should have:   * The size of the grid in the game | Y | The grid size is displayed alongside each previous game |
| 3 | Game Screen | | |
| 3.1 | When a game ends, the program should save to a database:   * The size of the grid in the game * The length of the line needed to win | Y | When a game ends, all of these data are saved to a database |

## Feedback Analysis

I obtained written feedback from a Maths teacher – an end-user involved in the questionnaire of the analysis stage. It demonstrates that I have achieved all of the core and additional objectives, which I set out to do in the analysis stage. According to the feedback, the variety of grid formats in the game (such as topology, grid size) was excellent and it encouraged users to be “adventurous”. Notwithstanding, the game is also beginner-friendly by supporting the use of adjacency maps and undoing moves. Furthermore, the single-player and multiplayer game modes were well received. Moreover, the feature of loading previous games and undoing moves allows users to explore different possibilities of moves they could have made. The cleaness and simplicity of the UI is satisfactory. However, the character input boxes for the: Side Length of Square Grid; Length of Line Needed to Win and Number of Moves Computer can Foresee Ahead were slightly tricky to use. Additionally, the names of each topology do not give much insight into what properties and shape they have.

## Overall Solution Effectiveness

Overall, the program I have created meets all of the requirements and objectives set out by the end-users in the analysis stage. It has a clean and user-friendly UI; the user can change various settings about the game; the user(s) can play a game in single-player and multiplayer mode; the user can view all of the previous games they have played (as well as information about these) and the user can load a previous game and cycle through each move in the game.

## Improvement

To improve, I could have made the character input boxes move user-friendly. A way to do this is highlighting their outline in a different colour when they are accepting input. This would mean that the user was aware when they can change the input, making the character input boxes less confusing.

Also, users may not understand what the properties of each topology are and how they affect the game. To address this, I could have either an image of each topology (or a self-intersecting depiction of what is possible in three dimensions) or a description associated with each option in the settings screen.

Furthermore, it be helpful to have the URL of the website, which explains the game, on the menu screen. This would mean that users have a better understanding of the game before they play it. It would also encourage them to be more adventurous when choosing their settings.

Moreover, if I had more time, I could have improved the design of the computer single-player algorithm. There is already a method to vary its difficulty by changing its recursion depth. Nonetheless, tt could assume that player 1 would make the best possible move instead of a random move. It would need a heuristic score function to enable it to evaluate what the best move is. This could, on a superficial level, be based on who has won or, on a more complex level, be based on how close each player is to making a winning line. This would align the algorithm with the design principles of the minimax algorithm. Nevertheless, this would have taken too much time to build into the project modelling in the analysis stage and follow through in the design and technical solution stages.

Overall, I have attained all of the objectives set out in the analysis stage but there are minor improvements and extensions that could be made.

# Appendix

## End-User Feedback

There was an excellent range of different formats of the game to try. The adjacency maps were brilliant for helping the nervous user at the start and I think these would encourage people to be more adventurous as they got used to playing the game. As well as different shapes of board, I liked the ability to alter the size of the board and the number of things in each winning line, as, again, these offered the opportunity to increase the challenge as the user became more experienced. The social and sharing aspect of playing against other players, as well as just playing against the computer, was brilliant. Being able to look at previous games encouraged me to try other options and being able to undo hesitant moves was also an encouragement to try out new ideas.

I found the program easy to use, though did need a hint as to how to alter the size of the board on the settings screen. A possible improvement might be to show a picture (where possible) or a description (where not) of what each shape might look like, or add a direct link to the place on the website where these were described.