[**https://youtu.be/iEXfgVlUQPI**](https://youtu.be/iEXfgVlUQPI)

[**Analysis 4**](#_c6m2dpawy6au)

[The Problem 4](#_lr1h4dg4l3w8)

[Objective 4](#_crv2p3bmotee)

[End Users 4](#_dhxnx5pwl9zz)

[Current Problem 5](#_li3akosfoblr)

[Key Themes 5](#_9yb2r5tbde1)

[General Knowledge 5](#_sbzhk0fqgewr)

[Logical Reasoning 5](#_8ody1fa2ngaj)

[Reaction Enhancing Features 6](#_egpbxscuvk9a)

[Measuring the Objectives 6](#_xnfu4on89go8)

[Survey Form for Objective 1 - General Knowledge 6](#_xu264ayt3fv3)

[Survey Form for Objective 2 - Logical Reasoning 7](#_d6630xtx8h94)

[Survey Form for Objective 3 - Reaction Enhancement 7](#_sujir8ox7l66)

[Game Stats for each Objective 7](#_fztn2rulgbrp)

[**Design 7**](#_gtl3faj8kq8a)

[Graphical User Interface 7](#_y1z4uwcw8gx5)

[Yeardle Main Screen 8](#_5uf0uhz7goe)

[Game Screen 12](#_5edjcbyeu633)

[Drop Zone 13](#_la2az6w1mp8p)

[Question Panel 15](#_yib2gkn7vqr)

[Score Board 15](#_hgv89f8apzqm)

[Bonus Ball 16](#_v2vywsjudkqj)

[UserQuestions 17](#_3eunbwhdcmjs)

[Object Oriented Design 18](#_6anjrof08fqp)

[Yeardle Class 21](#_a7r2x8qmfxxx)

[Properties 22](#_kzu23mj7f9f6)

[name\_select() Method 22](#_kbsdpxx8wk18)

[get\_user\_details() Method 23](#_xwqtox9rydsg)

[get\_leaderboard\_position() Method 23](#_vqyj8pexc68t)

[display\_stats() Method 23](#_wzqiglaf6j3j)

[add\_questions() Method 23](#_mglhcp4olays)

[display\_info(state) Method 23](#_716691fc7mb1)

[game\_on() Method 24](#_ypz2qu7hmgue)

[GameScreen Class 24](#_n6tvlprq3tt0)

[Properties 25](#_s4kl5vqba360)

[game\_on() method 26](#_pmbcy1yyczdn)

[start\_level() method 26](#_ltze9iirqlln)

[winner() method 26](#_86oxa8rla623)

[loser() method 27](#_uql8zc48psck)

[check\_answers() method 27](#_etu7bjohde9e)

[destroy\_widgets() method 28](#_sn48ta69one2)

[BonusBall Class 28](#_xqgacizcqn5u)

[Properties 28](#_1q0s1g9tin8u)

[reset() Method 29](#_egeg1cs217t)

[move() Method 29](#_knhqrme3nzw9)

[scored() Method 30](#_i63drbxkzy4b)

[YearWidget Class 30](#_j9pxbqekw5h6)

[Class Properties 31](#_r4y9uyp7dr3x)

[Properties 31](#_2uavn21pzme9)

[Drag and Drop Functionality 31](#_u33b49rto5x3)

[left\_mouse\_button\_clicked() method 32](#_tncd4h8qa7tx)

[mouse\_moving\_and\_left\_button\_clicked() method 32](#_1y82amhxz3hh)

[left\_button\_released() method 33](#_owghkwqi9mge)

[is\_overlap(x1,y1,width1,width2,x2,y2,width2,width3) method 34](#_dpexo46ag2fq)

[return\_to\_base() method 34](#_w6wz47rgv0to)

[DropZone Class 34](#_f1v9iywvrou)

[Properties 34](#_4kamcbwoulg3)

[show\_result() method 35](#_ttg1gv9bzju)

[QuestionPanel Class 35](#_2ioxdwlqwu3n)

[ScoreBoard Class 36](#_ogufk8dvrh6i)

[Properties 36](#_a5uce1eip3h)

[update\_scoreboard() method 37](#_glqaybg2kbyk)

[Mathematics Computations 37](#_eepta77cetr3)

[Projectile Motion of a Particle 37](#_fce38eaxi3dk)

[Database 39](#_he65yqt7h30z)

[**Technical Solution 39**](#_gcu5qaoujk54)

[Graphical User Interface 39](#_1d16p7lk1adu)

[The Main Yeardle Screen 40](#_vox6lz5ukf8c)

[Welcome Screen (state = 1) 43](#_hnfvjxxyh9c2)

[Game Over Screen (state = 2) 44](#_55f45q9jxmcl)

[Life Lost Screen (state = 3) 45](#_msafo8uwonoc)

[Completed Level Screen (state = 4) 45](#_bz5kk7gfc2a7)

[DropZone 46](#_3tmv3rkam5ji)

[Bonus Ball 48](#_wvb68rm9mo3u)

[Drawing the Bonus Ball on the Canvas 48](#_e33791z0msvq)

[ScoreBoard 51](#_egzuz3szaexl)

[YearWidget 52](#_ykvsaaydobq2)

[QuestionPanel 53](#_x6lzb493n3wb)

[GameScreen 54](#_6dvd9sr2q7wk)

[User Questions Screen 55](#_dojaclenm939)

[Behaviour Functionality 55](#_5dnku1gdnld6)

[BonusBall 56](#_54ds9ccgnuoz)

[move() method 56](#_o8emuyk2768h)

[reset() Method 57](#_idrej7g36bf8)

[scored() Method 58](#_57tgqmvspxpb)

[DropZone 59](#_6i4djhovgpmq)

[GameScreen 60](#_ys55dw6htnim)

[game\_on() method 60](#_ah0fdbf89y5q)

[start\_level() method 60](#_lkbm84sj6cbt)

[destroy\_widgets() method 61](#_q1hymcpei5r5)

[check\_answers() method 62](#_ecu6aevxpheo)

[winner() method 63](#_on87talw2k9g)

[loser() method 63](#_7wjocwlfmpoa)

[ScoreBoard 64](#_ly011oavxofm)

[Yeardle 65](#_rdrdzvfnmlii)

[State 1 65](#_b2numnjqy8ae)

[State 2 66](#_v9cy8eu52a62)

[State 3 67](#_kd5zki3wd9pd)

[State 4 67](#_rj1wvkzcmlpr)

[Other Methods 67](#_pkdgp6s78fn5)

[YearWidget 68](#_u0mwpyheg27e)

[Database 71](#_jb71q0wotm31)

[Creating the Questions Table 71](#_c8za58x3t8y1)

[Creating the User Table 72](#_khp3q6n0cl3h)

[QuizQuestions.py 72](#_qx8to1sqi4b2)

[generate\_questions\_from\_db() 73](#_fueqz6rr77lr)

[generate\_years() 73](#_kju1hl6dq121)

[get\_leaderboard() 74](#_4ftmnw98konh)

[update\_user\_score() 74](#_j07rc512ubf2)

[verify\_user() 75](#_rqq1cq25bek4)

[Data Gathering 75](#_e9zrgt5a250w)

[Full Code Implementation (Yeardle.py) 78](#_j9if7l19e9zk)

[**Testing 90**](#_kjazboqzywev)

[Testing Structure 90](#_4gl7zuqquw26)

[Iterative Testing 90](#_ibdpuhpw0tc0)

[Non Graphical Text Version (No Database) 90](#_5xr9foa6w5li)

[Non Graphical Text Version (With Database) 93](#_jro9kh67rig1)

[Graphical Non Object Oriented Version 94](#_f40sxhbtfjbw)

[Graphical OOP Version (unstyled) 97](#_as94y8vo771)

[Graphical OOP Version Fully Styled 98](#_92ul6wyzypt6)

[Post Development Testing 98](#_ja3eyjv74f08)

[**Evaluation 103**](#_s0nftrsohq0v)

[Educational Experience Through General Knowledge 103](#_cdcnwkp1unsr)

[Logical Reasoning 104](#_m5fzqolqw3nc)

[Reacting Enhancing Features 104](#_tqmiuyuk53b5)

[Summary 105](#_ndzz9hj86nc7)

VIDEO OF FINISHED PROGRAM: <https://youtu.be/iEXfgVlUQPI>

# Analysis

## The Problem

With the advent of the internet and social media people are finding more and more of their time being spent online engaging in activities that are often not in their best interests. Too much of social media and modern gaming is designed in a similar way to how gambling organisations design their products. In many cases the goal seems to be to exploit weakness in human psychology in order to keep people's attention online where they can be targeted by advertisers. Many people want to spend less time online but find it difficult to do so as the odds are stacked against them. My goal is to create an experience that does the opposite of the current options offered online.

## Objective

To create an experience via software that is educational and beneficial to the user whilst also being entertaining. To appeal to people of all abilities, backgrounds and age ranges.

To satisfy this goal I aim to create a graphical game that combines general knowledge, logical reasoning and reaction focused features. In speaking to friends and family members, I realised these are popular themes that people find engaging and beneficial.

## End Users

The end users of my game will consist of a wide variety of people of all ages. The intention is to create a game that has features that will appeal to a large group of people. The game will need to be accessible to people of all abilities. People with a high level of interest in General Knowledge should find the experience engaging but people with much lower interest should also be able to take part. People who worry about their cognitive functioning deteriorating as they get older should be able to benefit as well.

## Current Problem

There are no games that combine such a wide range of topics such as general knowledge, logical reasoning and reflexes. That is why Yeardle is needed and why it's not like any other game. It will help teach users skills that they may not have had before such as fast reflexes and more general knowledge skills. They can use these skills in other aspects of life also.

## Key Themes

### General Knowledge

I want to ensure a wide range of general knowledge topics but with a consistent method of answering each question. To meet this goal I decided to choose historical events and the years that they happened.In order to have the broadest appeal, the people I discussed the idea with seem to focus on the following key areas:

* Politics and History
* Music
* Movies and TV
* Celebrities
* Sports

Typical questions could be of the form:

* “What year did the titanic sink?” (Answer: 1912 , Category: Politics & History).
* “What year did Ed Sheeran have a number 1 with ‘Shape of You’” (Answer: 2017, Category: Music)
* “What Year was the movie ‘Mean Girls’ released?” (Answer 2004, Category: Movies)
* “What year was Kim Kardashian Born?” (Answer: 1980, Category: Celebrities)
* “What year did Brazil beat Italy 4-1 in the world cup final?” (Answer: 1970, Category: Sports)

### Logical Reasoning

To meet this goal I wanted to ensure that even if some of the general knowledge questions proved too challenging, that the game could still be played by applying logical reasoning. In discussing with my target audience it appears a very popular way of achieving this would be through a ‘mastermind’ style of game play.

In the case of my game the approach would be to display 4 questions and also display 10 possible answers. The user would then assign 4 of the answers to each of the questions and the program would then provide feedback on how accurate the answers were.

The program will display a black circle for every answer that is correct and positioned against the correct question and display a white circle for every answer that is correct for one of the questions but has not been positioned next to that question. The user can then use logical reasoning to improve on the attempts in subsequent rounds

### Reaction Enhancing Features

To meet this goal I will include a bonus feature into the game that involves fast reflexes. The idea is to have a bonus ball fly across the screen during game play and if the user is fast enough to click on it, the game will be made simpler. In addition, this will be an opportunity to introduce some A-Level standard mathematics into the game. The motion of the bonus ball should look realistic so a user can correctly anticipate the correct location to click the screen. This will require use of the mathematical motion equations (SUVAT).

## Measuring the Objectives

I need to ensure the each of the 3 key objectives of the project are measurable

| # | Objective | How it will be Measured |
| --- | --- | --- |
| 1 | General Knowledge | User Survey / Game Stats |
| 2 | Logical Reasoning | User Survey / Game Stats |
| 3 | Reaction Enhancement | User Survey / Game Stats |

The user survey will be given to a select group of users after they have played the game a set number of times. The final survey will evolve based on the implementation of the game - however a reasonable attempt at the surveys can be attempted at this Analysis stage

### Survey Form for Objective 1 - General Knowledge

1. How easy hard did you find the general knowledge questions (1- Easy/ 5 Hard)
2. Did you feel your knowledge was improved the more you played the game (1-Not at all, 5 - Significantly So)
3. How interesting did you find the questions (1-Boring, 5 - Highlighly Engaging)
4. Which was your favourite Category of Question (Sport, Celebrities, History & Politics, Music, Movies & TV)

### Survey Form for Objective 2 - Logical Reasoning

1. What tactics/ different methods did you use when figuring out which years belong to each question?
2. Do you think you were given enough attempts to get the correct answer?
3. Do you think you were given enough time to get the correct answer?
4. Did you use any maths to solve the problem?
5. Is there anything that can be improved with this game?

### Survey Form for Objective 3 - Reaction Enhancement

1. How old are you( age may affect people's reaction times)
2. Did you get better at hitting the bonus ball with practice
3. Did you find that this feature of the game helped your reaction time in day to day activities?
4. How hard did you find it to click the bonus ball in time
5. Is there anything that can be improved with this game?

Survey to be designed

### Game Stats for each Objective

It is well known that Surveys do not always present an accurate picture of a person's experience. This could be to do with the design of the Survey or the fact that people are not always fully aware of how the experience has actually affected them. One method to solve this issue is to capture information from the actual software itself. For example the software could record the times taken to complete different levels and the accuracy of a specific players’ answers over time as they play the game more and more. In terms of reaction times the game could record the frequency that a user captures the bonus ball and track if this improves the more the user plays the game. This ability will be beyond the scope of this project but is something that should definitely be included in a future version to assist in the further development of the game.

# Design

I decided early on that I wanted to build a graphical user interface instead of a command line interface as this would make the experience far more engaging for the end user. It was also the only way I could have included the video gaming features highlighted previously. In addition to this, I decided to use an actual database (Sqlite3) to store all the information that the system would be using.

## Graphical User Interface

The GUI will consist of 8 key components

1. Yeardle Main Screen - this will display information about the current user, display the leaderboard, Welcome screen and information screens between levels. It will also provide the link to both the Game Screen and the User Questions screen.
2. Game Screen - this will be the main game screen responsible for positioning the rest of the visual components
3. Question Panel - Will display a list of 4 questions (events that occurred in specific years)
4. Drop Zone - This is the area of the screen where users will be ‘drop’ their answers - this component will also display a Done button for when they wish to submit their answers and a results panel to display the result of their guesses
5. YearWidget - YearWidget will be a rectangle containing a possible correct Year - the Yeardle Main Screen will display up to 10 Year Widgets and these Year Widgets will be ‘draggable’
6. Bonus Ball - This will appear randomly within the Yeardle Main Screen and be ‘clickable’ meaning that a user will gain a bonus if they manage to click the Bonus Ball when it appears. The Bonus Ball will be animated so that it flies across the screen according to the laws of Physics
7. Scoreboard - This will display the current remaining time and the current score and number of remaining lives for the current player
8. User Questions - This will be a popup that will allow the user to add new questions/events to the database

### Yeardle Main Screen

The Main Yeardle Screen will be responsible for displaying the Welcome Screen, displaying the current leaderboard, allowing a user to enter a username or create their own questions to add to the database. The layout will look as follows:

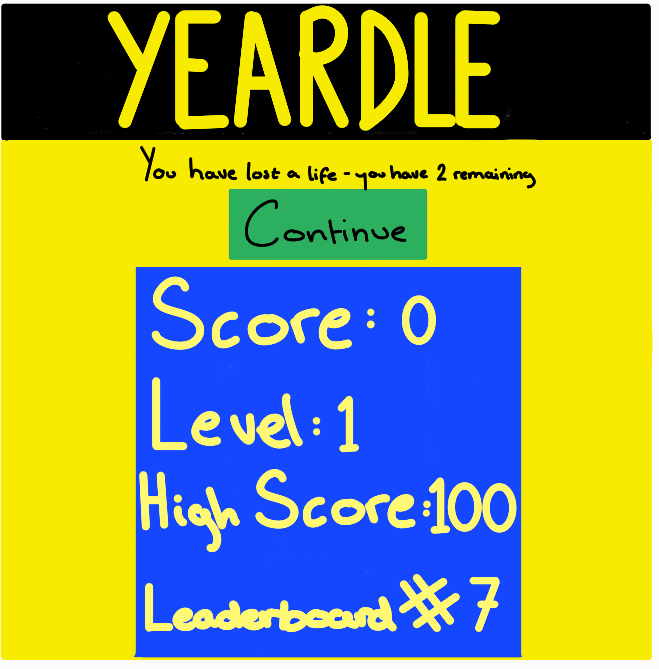


If the User clicks the Game On button they will be taken to the Game Screen (see below)

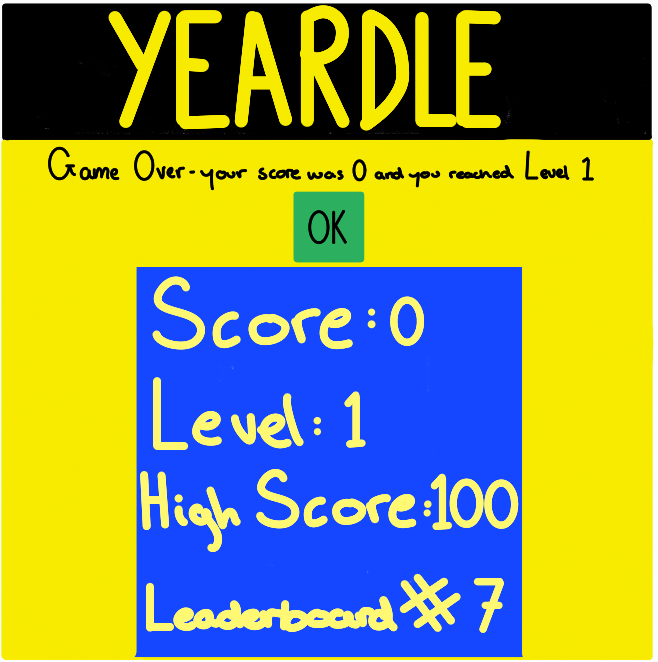
If the User clicks the Add Questions button they will be taken to the User Questions screen (see below)

In addition to these key features this screen will also be responsible for providing the user with information and options in the following 3 scenarios:

1. The User Loses a life - this will display the current stats for the player and allow them to Continue the game in progress (example below)



1. The Game Over screen - in the event the user loses all their lives they will be presented with the Game Over screen (example below)



1. Successful Level Completion - if the user successfully completes the current level they will be presented with the Level Completion screen (example below)

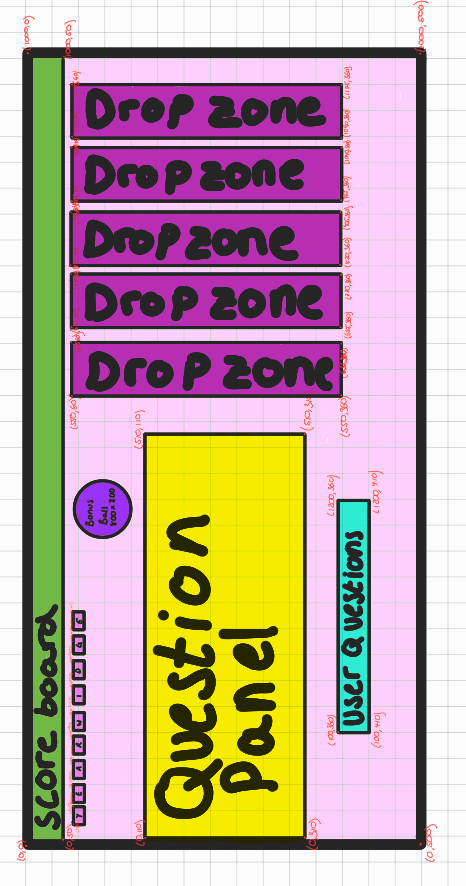


### Game Screen

The Game screen will be 1000 pixels wide by 500 pixels deep. This screen will be responsible for placing all the other GUI components according to the following coordinate system (all values in pixels). Note (0,0) represents the top left hand corner of the main Yeardle Screen)

* Scoreboard - this will be placed at (0,0) and have a width of 1000 pixels and a height of 50 pixels
* Question Panel - this will be positioned at (0,110) and have a width of 510 pixels and a height of 200 pixels.
* Drop Zone - there will be up to 5 of these displayed during the game - they will each have a width of 80 pixels and a height of 300 pixels. The first Drop Zone will appear at position (550,60) and each of the additional Drop Zones will be placed 82 pixels to the right of the previous zone (eg (632,60), (714,60) etc…)
* Year Widgets - At the start of each round there will be 10 of these displayed on the screen - the first will be placed at (10,60) and then each subsequent will be placed 40 pixels to the right of the previous one (eg (50,60), (90,60) etc…)
* Bonus Ball - this will be an animated component that will appear randomly within the Yeardle Main Screen so it will have no fixed position. It will be a square of 80x80 pixels.
* User Questions will be a pop up screen that will be accessed from the Yeardle Main Screen. It will have a width of 1100 pixels and a height of 500 pixels.

Below is an overview of the Game Screen with the components positioned as described above:

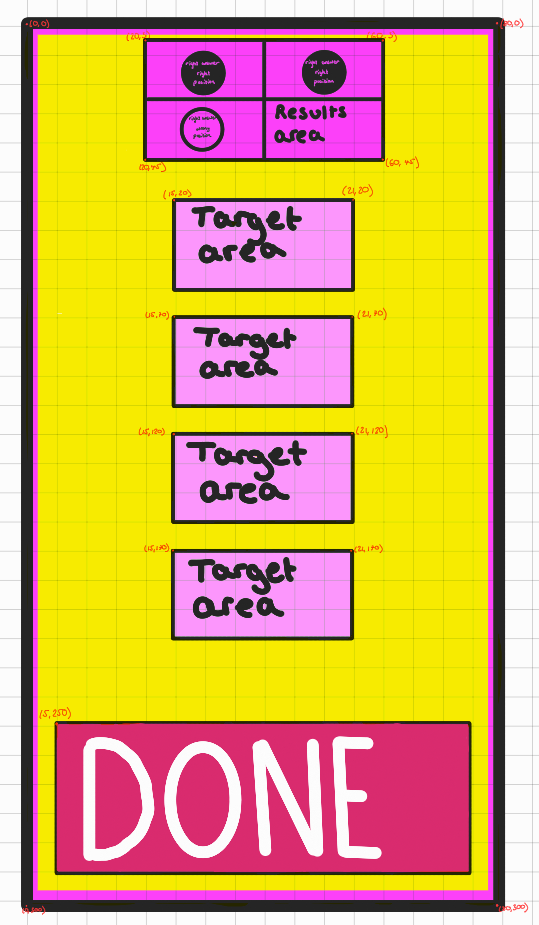


### Drop Zone

The drop zone has a width of 80 pixels, a height of 300 pixels and will consist of 3 GUI components:

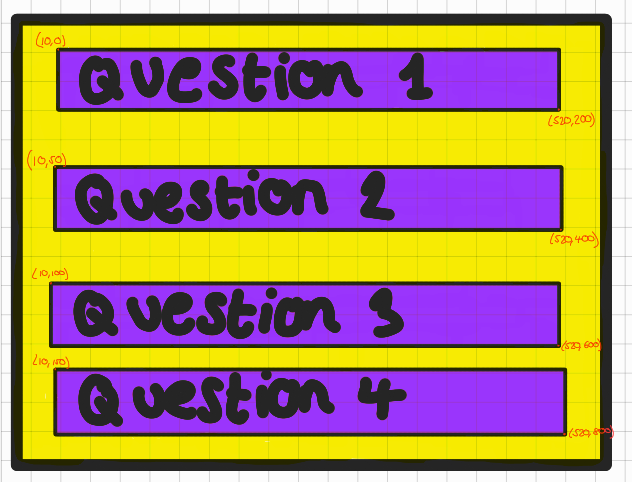
1. Results area - this will be 40x40 pixel square placed at position (20,5) within the DropZone area. The purpose will be to display the result of each player turn - this square will be further divided into 4 quadrants. Each quadrant will display either a black circle, a white circle or nothing. A black circle will represent the fact that the user got one of the questions correct and placed the answer in the correct target position. A white circle will represent the fact that the user got one of the questions correct but placed the year in the wrong target area.
2. Targets Area - this will consist of 4 rectangles that the user will place their guesses/answers (Year Widget) into. The first will start at (15,20) and have a width of 6 characters. The next one will be 50 pixels below etc.. (so at (15,70) then (15,120)...)
3. Done Button - this will be a button that only becomes available once the user has placed a Year Widget into all 4 of the Targets above. When this button is clicked the system will calculate the result for the user and display in the Results area above. The done button will be placed at the bottom of the the drop zone at location (5,250)

The design for this can be seen below:



### Question Panel

The first question will be placed at (10,0) then second at (50,0) the third at (100,0) and the 4th at (150,0) within a frame of size 510x200 pixels. This is shown in the screen design below:



### Score Board

The scoreboard will be 1000 pixels wide by 50 pixels high and consist of 6 components as follows:

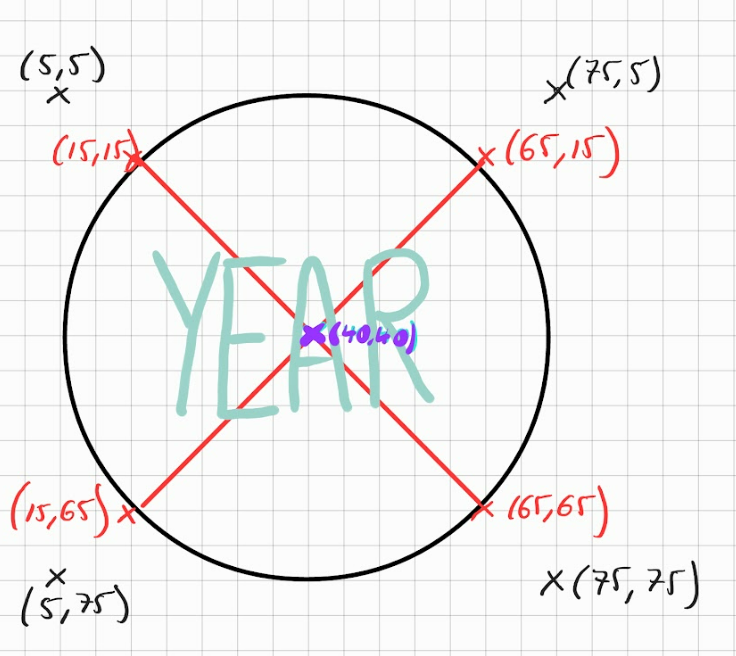
* Countdown timer at coordinate (5,5)
* Score at coordinate (60,5)
* Title (ie “YEARDLE”) at coordinate (400,4)
* User Details at coordinate (700,5)
* Remaining Lives at coordinate (850,5)
* Current Level at coordinate (950,5)

The scoreboard itself will be position at coordinate (0,0) within the main Game Screen.



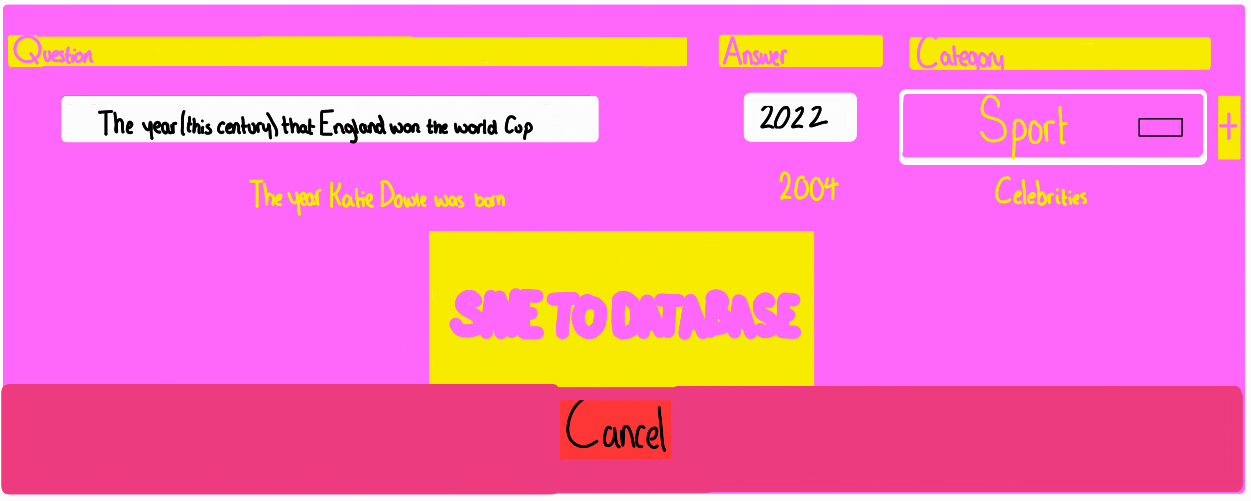
### Bonus Ball

I decided to create a purple circle with the word YEAR in cyan and with a red diagonal cross through it. This needed to be designed on a square grid. I sketched it out on an ipad below - each square represents 5 by 5 pixels. The black coordinates represent the coordinates of the circle (the smallest square that the circle is to be drawn in). The red coordinates represent the start and end points of the red cross. The purple coordinate represents the centre of the image - this is where the word “YEAR” is to be placed.



### UserQuestions

The UserQuestions screen will look as follows



There will be 3 headings at the top of the screen - Question, Answer, Category

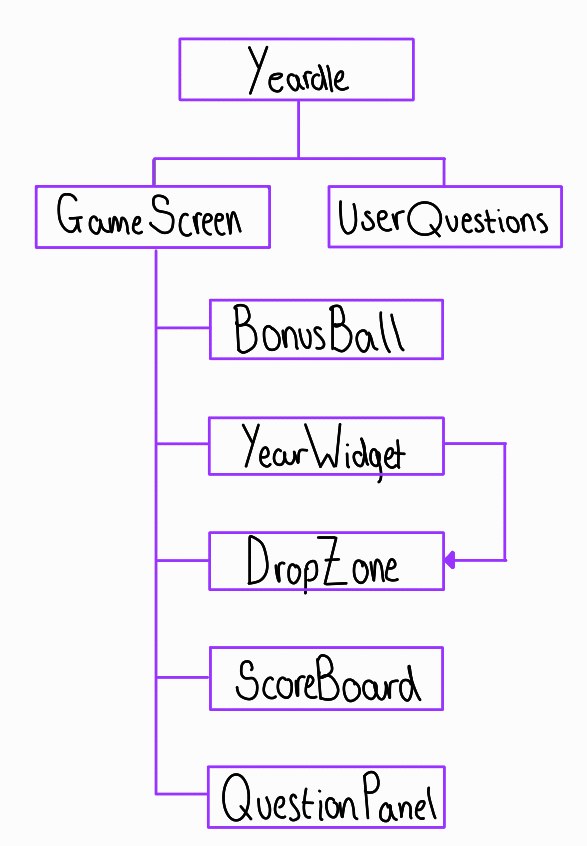
Below that there will be 3 GUI elements - 2 of these will be text entry fields (for the Question and Answer) and the last item will be a drop down list of options for category - this will include the following:

* Celebrities
* Sport
* Movies & TV
* Music
* Politics and History

The user can add as many questions as they like by clicking a + button. When they have finished they can save all the questions to the database by clicking a button called Save to Database. The user can also Cancel to go back to the main Yeardle screen.

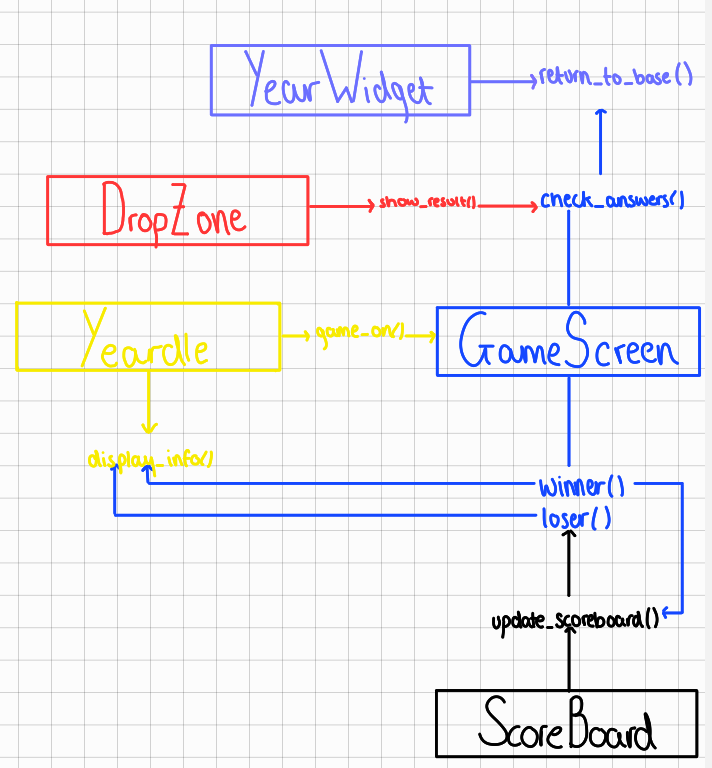
## Object Oriented Design

The system will follow object oriented design principles. There will be 8 Classes organised as follows:



The main Yeardle class will contain references to the GameScreen and the UserQuestions class. The GameScreen class will contain references to the BonusBall, YearWidget, DropZone, Scoreboard and QuestionPanel classes. The YearWidget will also contain a reference to the DropZone class so that they can communicate their relative positions.

The Classes will be designed so they are self contained however there will be a need to communicate between classes. The key areas that have been identified are shown below:



The Yeardle class will communicate with the GameScreen class in 2 ways - the Yeardle class will be responsible for calling the game\_on() method of the GameScreen class. Likewise the GameScreen class will have to call the display\_info() method of the Yeardle class when a player either Wins a level or loses a life - this will be done via the GameScreen winner() and loser() methods.

The GameScreen will also communicate with the Scoreboard by calling its update() method after a player wins a game. Likewise the ScoreBoard class will communicate with the GameScreen class when a player runs out of time (the ScoreBoard class will be responsible for keeping track of the elapsed time during a game)

The DropZone class will communicate with the GameScreen class whenever a player hits the Done button of the DropZone instance. In this case the GameScreen will be responsible for computing the result of the player's guess. The GameScreen will also communicate with the YearWidget class at this point in order to return all the YearWidgets back to base after each player attempt.

All other functions within the game will be handled internally within each of the Classes.

Below is a Unified Modelling Language representation of the classes in the system and a full description of the methods and properties

### Yeardle Class

| **ClassName: Yeardle , Inherited from Tk** |
| --- |
| **Properties**  user: string  high\_score: int  high\_level: int  **Class References**  game: GameScreen  userquestions: UserQuestions  **GUI Properties**  info\_screen: Tk.Frame instance  maintitle: Tk.Label  message: Tk.Label  stats: Tk.Label  playbutton: Tk.Button  addquestions: Tk.Button  cancel: Tk.Button  unameentry: Tk.Entry  leaderboard: Tk.Listbox |
| **Methods**  name\_select()  get\_user\_details()  get\_leaderboard\_position()  display\_stats()  add\_questions()  display\_info(state)  game\_on() |
| **Bindings**  Item Selected from Leaderboard → nameselect() |

#### Properties

* user: username of current player
* high\_score: high score of current player
* high\_level: highest level achieved for current player

GUI Properties

* info\_screen: this will be a container for the below elements
* maintitle: this will simply display the word YEARDLE in large font
* message: Will display the different messages to the user at different times in the game (ie at start of game or when user loses a life or completes a level)
* stats: Will display stats for current user (eg score, level, leaderboard position)
* playbutton: A button that will start the game when clicked
* addquestions: A button that will launch the UserQuestions screen to allow users to add their own questions
* cancel: This will be displayed with the UserQuestions screen to allow player to cancel if they have pressed the button by accident
* unameentry: A text entry box to allow players to enter a username
* leaderboard: A listbox that will contain the leaderboard - a list of all players who have played before ranked by score

Class References

* game: an instance of the GameScreen class
* userquestions: an instance of the UserQuestions class

#### name\_select() Method

This will be called when the user selects a player from the leaderboard. The username entry box will now be replaced with the username that has just been selected

#### get\_user\_details() Method

This method will check if the username exists in the database and if so it will retrieve the current high score and high level for the user. If not it will add a new user to the database. This will be done via the database functions (see Database design section)

#### get\_leaderboard\_position() Method

This method will retrieve the current leaderboard position for the current user

#### display\_stats() Method

This will display information regarding the current player - it will show the following:

Score - the current score of the game in progress

Level - the current level of the game in progress

High Score - the highest score for the current player

Leaderboard# - the current leaderboard position for current player

#### add\_questions() Method

This will create an instance of the UserQuestions class and place it on the main screen

#### display\_info(state) Method

This will be responsible for displaying the different types of information to the user. There will be 4 states supported and different information will be displayed depending on the state

State 1: Welcome Screen

This will display a welcome message and the current leaderboard. It will also provide an Entry text box for the user to create a new user name. This screen will have 2 buttons - one button will be the Game On button to start the game (*see game\_on() method*) and the other will be the Add Questions button to allow a user to add their own questions to the system (*see add\_questions() method*). The user will either be able to select their username from the current leaderboard or if this is their first time playing they can create a new username.

State 2: Game Over Screen

Once a game has been played the user will be shown this screen. They will shown a message that displays the score they achieved and the level they got to. If they have achieved a new high score they will also be informed of this. This screen will also display the current stats for the player (see *display\_stats()* method)

Finally this screen will have an OK button which will take the user back to the Welcome Screen (State 1)

State 3: Lost a Life Screen

When the user loses a life this screen will be show. They will be shown a message and told the number of lives remaining. They will also be shown the current game stats (see display\_stats() method). A Continue button will be displayed which will take the user to the start\_level() method of the GameScreen class.

State 3: Completed Level Screen

When a user successfully completes a level this screen will be show. They will be shown a message informing them of the level they have completed and will be shown the current game stats (see display\_stats() method). A Continue button will be displayed which will take the user to the start\_level() method of the GameScreen class.

#### game\_on() Method

This will first check that the current username is acceptable - it will have to be at least 3 characters long and no more than 20. The username can also not contain any spaces. If the username is not acceptable then the player will have to retype and try again. If the username is acceptable then a new instance of GameScreen will be created and the game\_on() method of the GameScreen will be called.

### GameScreen Class

| **ClassName: BonusBall , Inherited from Tk.Frame** |
| --- |
| **Properties**  in\_play: Boolean  attempt: int  questions: String[]  guesses: Integer[]  answers: Integer[]  years: Integer[]  score: Integer  lives\_remaining: Integer  current\_level: Integer  time\_allowed: Integer  bonus\_ball\_frequency: int  max\_bonus\_balls: int  **Class References**  scoreboard: ScoreBoard  questionpanel: QuestionPanel  bonus\_ball: BonusBall  dropzones: DropZone[]  yearwidgets: YearWidgets[]  parent: Reference to the Yeardle class  user: reference to the User from Yeardle class  high\_score: reference to high\_score from Yeardle class  high\_level: reference to high\_level from Yeardle class |
| **Methods**  destroy\_widgets()  game\_on()  start\_level()  winner()  loser()  check\_answers() |
|  |
|  |

#### Properties

* attempt - this will track how many attempts have been made in the current round
* questions - This will be a list of strings containing the 4 questions for the current round. This should be obtained by making a call to the database (see database section for more details)
* answers - this will store the correct answers for the 4 questions - this should also come from the database. It will be a list of 4 integers
* guesses - this will track the guesses that the user has made during this round - it will be a list of 4 integers
* years - this will be a list of 10 integers each being a 4 digit year. It will contain the 4 correct years and 6 ‘dummy’ years that are close to the correct years. This will be generated in the database module so please refer to this for more details
* score - the current score
* live\_remaining - the number of lives currently remaining
* current\_level - the current level within the game
* time\_allowed - the time allowed to complete current level
* bonus\_ball\_frequency - the frequency in seconds that on average each bonus ball is expected to appear
* max\_bonus\_balls - the maximum number of bonus balls allowed on current level

Class Reference

* user - a reference to the user property of the parent class (Yeardle)
* high\_score - a reference to the high\_score property of the parent class (Yeardle)
* high\_level - a reference to the high\_level property of the parent class (Yeardle)
* scoreboard - this will be an instance of the scoreboard class
* questionpanel - this will be an instance of the questionpanel class
* bonus\_ball - this will be an instance of the bonus\_ball class
* dropzone - this will be a an instance of the DropZone class
* yearwidgets - this will be an array of 10 YearWidget class instances

#### game\_on() method

This method will set the various game properties to the default settings - this may change but the setting initially will be as follows:

* score = 0
* live\_remaining = 3
* current\_level=1
* time\_allowed=300
* bonus\_ball\_frequency = 10
* max\_bonus\_balls = 6

Once these have been set the start\_level() method will be called

#### start\_level() method

Set the following properties:

* in\_play = True
* attempts = 1
* questions = set of 4 randomly generated questions from database
* answers = answers for the above questions
* years = set of 10 years - 4 correct , 6 dummies

Then create instance of the key classes

* Create a new empty array of DropZones
* create a DropZone instance, place on screen and add to array
* create a BonusBall instance
* create a ScoreBoard instance and place on screen
* create a QuestionPanel instance and place on screen
* create a new empty array of YearWidgets
* generate 10 YearWidget instances based on the **years** property, place each of them on screen and add to the YearWidgets array. Each YearWidget will contain a link to the current DropZone

#### winner() method

Will be called when the current player successfully completes a round. A points score will be calculated as follows:

points = current\_level \* (6-attempts) \* 20

This means a player will get a higher score as the levels move up and they will also get a higher score if they complete the round in fewer attempts. These points will then be added to the player score

The current\_level should then be increased by 1 and the level should be made more difficult, this will be done by reducing the time allowed for the next level, decreasing the bonus ball frequency and reducing the amount of bonus balls by 1 (eventually the player will have no more bonus balls as they progress to higher levels).

The scoreboard should then be updated to reflect the new score and the parent class display\_info() method should be called with state = 4

#### loser() method

Will be called when the player loses a life. This will take 1 away from the number of lives and check if there are no lives remaining. If there are no lives remaining then it will call the Yeardle class display\_info() with state = 2 (Game Over) else it will call the Yeardle class display\_info() with state = 3 (Lost a Life)

#### check\_answers() method

This will be called from the DropZone class when the Done button of that class has been clicked. It will then be responsible for calculating the result of the user guesses by comparing them to the actual answers. This is the code that will represent the ‘Mastermind’ component of the game. The standard procedure is to step through each of the 4 elements in the guesses array and compare it to the elements in the answers array. I will start with an empty string variable **results**.If the elements match then we add a “1” to this string, if the guess element is contained in the answers array but the position is not the same then i’ll add a “0” to the results string. A result string of “110” would represent a situation where 2 of the guesses are correct and in the correct position and one of them is correct but in the wrong position - this would also mean that one of the guesses is wrong. I then return this string to the DropZone instance - it will then be responsible for displaying the result in a graphical way.

At this point I'll also check if the player has got all 4 answers correct , ie result = “1111” in this case I will call the **winner()** method. I can also check if the attempts has gone beyond 5 (the maximum allowed attempts) - if this is the case then I will call the **loser()** method.

The final thing to do is to create a new DropZone instance and move all the YearWidgets back to base

#### 

#### destroy\_widgets() method

The GameScreen class is responsible for creating several GUI components. These components will have to be destroyed between levels - this method will take care of this process.

### BonusBall Class

| **ClassName: BonusBall , Inherited from Tk.Canvas** |
| --- |
| **Properties**  bonus\_start: Boolean  years: List  answers: List  average\_bonus\_ball\_time: int  vertical\_acceleration: Float  flight\_time: Float  start\_velocity: Float  angle\_to\_horizontal: Float  start\_x: Integer  start\_y: Integer |
| **Methods**  reset()  move()  scored() |
| **Bindings**  Left Mouse Button → scored() |
|  |

#### Properties

The Bonus Ball will have the following properties:

* **Bonus\_Start** -- this is a boolean value which will determine if the Bonus Ball is ‘active’ - when this is True the ball will move across the screen
* **Years** - This is a list of the current years (from the Yeardle class)
* **Answers** - List of correct years (from the Yeardle class)
* **average\_bonus\_ball\_time** → this is an integer that represents the average period of the bonus ball activity. For example if this was set to 30 then we would expect the Bonus Ball to be activated on average every 30 seconds.
* **flight\_time** → this is a float that represents the time (in seconds) that the ball has been in flight - this will be used in the SUVAT equations (please see Projectile Motion of a Particle section) to compute the displacement of the bonus ball at a specific point in time
* **vertical\_acceleration** - This will probably need to be fixed at -9.81 to represent an estimate of a particle's acceleration due to gravity (again this is used in the SUVAT equations)
* **start\_velocity** - the initial velocity of the Bonus Ball
* **angle\_to\_horizontal** - the launch angle (in degrees) of the Bonus Ball once activated
* **start\_x** - the x coordinate of the start position of the bonus ball - this will be either 0 or 1000 and will thus determine if the ball will travel from right to left or left to right
* **start\_y** - the y coordinate of the start position of the bonus ball

#### reset() Method

This method will be responsible for setting up the initial conditions of the bonus ball - ie:

1. Set the **bonus\_start** property to False and remove the Bonus Ball from the screen. This will put the bonus ball in a dormant state which means it will not be displayed or animated on the screen
2. Set the **flight\_time** property to 0
3. Set the **start\_y** value to a random number between 150 and 300
4. Set the **start\_x** value to either 1000 or 0 - this will be determined randomly
5. Set the **angle\_to\_horizontal** to be a random number between 1 and 60 degrees
6. Set the **start\_velocity** to be a random number between 3 and 7

#### move() Method

The move method will be responsible for moving the bonus ball object across the screen. It will do so by computing a new position every fraction of a second. The mathematics behind the position can be seen in the section Projectile Motion of a Particle under the Mathematical Computations heading. The move method will be recursive and call itself repeatedly at a set interval X milliseconds. Each time this will be called there will be 2 things that can happen

1. The Bonus Ball will be dormant so I need to determine randomly if it should be launched. I also need to check if the maximum number of bonus balls has been exceeded - in which case I will keep the dormant state for the remainder of the level
2. The Bonus Ball will be moving

If the Bonus Ball is dormant then I will need to generate a random number based on the average\_bonus\_ball\_time and the call interval of the move method X. Since X will be measured in milliseconds I know that X=1000 would be 1 second. If the average\_bonus\_ball\_time was set to say 30 then I would need to generate a random number between 0 and 30 each time the move function is called - then we would compare this to a set number (eg 5) and if they match then I would put the bonus ball into an active state (by setting its bonus\_start property to True). Note in general I would have to generate the random number to be between 0 and **average\_bonus\_ball\_time \* 1000/X.**

The second thing that the move() method will be responsible for is to actually move the ball. So I would need to check if the bonus\_start has been set to true and then compute a new position for the ball. I would do this by increasing the flight\_time by a small increment and then computing the horizontal and vertical displacement of the particle for the specific flight time. I would then add this displacement to the (start\_x, start\_y) initial coordinate and move the Bonus Ball image to this new location. The ball would then continue moving until one of the following occurs:

1. The ball reaches the end of the screen (either left right or bottom)
2. The ball is successfully ‘hit’ by the user (by using left mouse button to click it)

Once either of these 2 condition have been met the Bonus Ball will become dormant again and the cycle repeats

#### scored() Method

This method will be called if the left mouse button is clicked anywhere on the BonusBall image. If this happens then one of the Year Widgets that does not contain a correct answer needs to be removed. The first thing to do is find all the Year Widgets that do not contain a correct answer and then select one at random to be removed. I then call the reset() method on the Bonus Ball to put it back into the dormant state.

### YearWidget Class

| **ClassName: YearWidget , Inherit from Tk.Label** |
| --- |
| **Class Properties**  POS\_COUNT: Integer  COLOURS: String[]  **Properties**  year: Integer  dropzone: DropZone  original\_x: Integer  original\_y: Integer  in\_position: Boolean  colour\_index: Integer  mouse\_x: Integer  mouse\_y: Integer |
| **Methods**  left\_mouse\_button\_clicked()  mouse\_moving\_and\_left\_button\_clicked()  left\_button\_released()  right\_mouse\_button\_clicked()  is\_overlap(x1,y1,width1,height1, x2,y2,width2,height2) - returns Boolean  return\_to\_base() |
| **Bindings**  Left Mouse Button → left\_mouse\_button\_clicked()  Right Mouse Button → right\_mouse\_button\_clicked()  Left Mouse Button Dragging → mouse\_moving\_and\_left\_button\_clicked()  Left Mouse Button Released → left\_button\_released() |
|  |

#### Class Properties

POS\_COUNT - this class level property is an integer that will keep a count of how many instances have been successfully placed into the Drop Zone target areas.

COLOURS - this will be a string array containing 3 colours - “Yellow”, “Red” and “Green”. This will be used in the right\_mouse\_button\_clicked method to change the colour of the current instance

#### Properties

* year - stores the actual year associated with the current instance
* original\_x - when an instance is created this will store the x coordinate of it initial placement
* original\_y - when an instance is created this will store the y coordinate of it initial placement
* dropzone - a link to the current dropzone instance - this will be required to determine if the current YearWidget instance has been placed on a target within the drop zone
* in\_position - if the current instance has been successfully placed in a drop zone target then this property will be set to True, else it will be False
* colour\_index - this will be an integer that represents the index of the class level COLOURS array - this will be used to set the background colour of the current instance
* mouse\_x: this will be an integer that stores the current x coordinate of the mouse pointer
* mouse\_x: this will be an integer that stores the current y coordinate of the mouse pointer

#### Drag and Drop Functionality

The following methods all relate to drag and drop functionality. The game is expected to be played on a windows computer and on such a device there is a specific way a user would carry out a drag and drop procedure:

1. The first thing a user would do is to click on a visual item that needs to be moved - they would do this using the left mouse button. In this case a computer program needs to detect if the left mouse button has been clicked on the visual item.
2. The second thing a user would do is to ‘drag’ the visual item - they would do this by moving the mouse to the desired location whilst keeping the left mouse button pressed. In this case a computer program would have to detect that the mouse if moving whilst the left mouse button is being held
3. The final stage is the drop stage - this is when the user releases the left mouse button in order to ‘drop’ the visual item at the current location.

##### left\_mouse\_button\_clicked() method

This will be called when the left mouse button is clicked on the current year widget instance. This method will simply set the mouse\_x and mouse\_y properties to the current position of the mouse pointer.

##### mouse\_moving\_and\_left\_button\_clicked() method

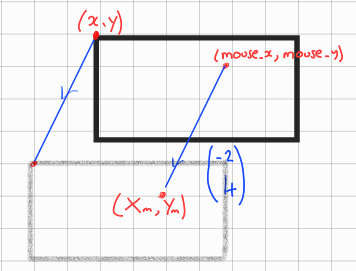
This will be called when the left mouse button is held down and the mouse is being moved at the same time. This will mean that the current YearWidget instance is being dragged so I need to calculate the new location of the YearWidget and then move the YearWidget to this location. This method will be continuously called until the left mouse button is released.

In order to compute the new location of the current instance I will need to work out the displacement vector for the mouse move since the left mouse button was first clicked. The original mouse coordinate was (mouse\_x, mouse\_y) so if I call the current mouse coordinate (X,Y), then the displacement vector is given as (X-mouse\_x, Y -mouse\_y). Now I need to add this displacement vector to the current location of the YearWidget (lets call this (x,y)) - so I would compute the new location as:

new\_x = x + X - mouse\_x

new\_y = y + Y - mouse\_y

The YearWidget instance would then be moved to this new location. This is shown using an example in the below diagram (the black rectangle is the current location of the widget and the grey rectangle is the new position based on the mouse movement):



##### left\_button\_released() method

This method will deal with the final stage of ‘drag and drop’ which is the drop stage. The first thing I need to do is calculate the top left hand corner of the YearWidget - I will do this the same way as in the previous method.

new\_x = x + X - mouse\_x

new\_y = y + Y - mouse\_y

Then I would have to step through each of the 4 targets in the drop zone and check to see if the current Year Widget overlaps it. To do this I need to calculate the width of the current year widget and the width and top left hand corner of each of the targets. To then calculate if there is an overlap I would call the is\_overlap() method

If there is an overlap then I will need to ‘lock’ the Year Widget into position. This will involve the following steps:

1. Add 1 to the class level **POS\_COUNT** variable
2. Set the **in\_position** property to True
3. Place the widget at (new\_x, new\_y) location
4. If POS\_COUNT == 4 then enable the Done Button from the DropZone instance, otherwise disable it

If there is no overlap then I would need to call the return\_to\_base() method to move the widget back to its original position

#### is\_overlap(x1,y1,width1,width2,x2,y2,width2,width3) method

This method will compute if 2 rectangles are overlapping and return True if they are or False if they are not. The 2 rectangles will be represented as a top left coordinate and a width and height value. The method would then compute the 4 vertices of the first rectangle and then check to see if any of these points lies within the 2nd rectangle. If any point does then the method should return True immediately. If none of the points from the first rectangle lies within the 2nd then the method should calculate all the vertices of the 2nd rectangle and check to see if any of these lie inside the 1st rectangle.

#### return\_to\_base() method

Each YearWidget will start out at a specific location on the main screen. There are 2 instances when a widget will need to be returned to this location. The first is at the start of each new attempt by the user. The second is if the user attempts to drag an item and does not drop it in a specific target or if the user drags an item from a target area. In any of these cases I will need to move the location of the widget. In this case I would need to check if the widget was currently in a target location (I could do this by checking the in\_position property) - in this case I would need to take 1 away from the class level POS\_COUNT property (since this variable will keep track of how many Year Widgets are currently in target positions). I would then need to simply place the widget back in its original location **(original\_x, original\_y)**

### DropZone Class

| **ClassName: DropZone , Inherited from Tk.Frame** |
| --- |
| **Class References**  parent: Yeardle  **GUI Properties**  results: Tk.Canvas  done\_button: Tk.Button  targets: Tk.Label[] |
| **Methods**  show\_result() |

#### Properties

Class References

* parent - this will be a link to the main Yeardle class - this will be required so that I can have the main class compute the result of the user guess

GUI Properties

* results - this will be an instance of the Canvas object within Tkinter. It will be used to draw black and white circles that represent the results of a user guess
* done\_button - this will be an instance of the Button object within Tkinter. When clicked it will call the show\_result() method. This button will only be enabled when all 4 user guesses have been placed within the DropZone.
* targets - this will be a list of Label objects (again from the Tkinter library) - each one represents an area of the drop zone where a Year Widget instance can be placed

#### show\_result() method

This method will call the check\_answers() method from the main Yeardle class which will return a string containing the result of the current user guess. For example “100” will mean that 1 of the guesses is correct and in the correct position and 2 of the guesses are correct but in the wrong position - the final guess is incorrect. Once the result is returned this method will be responsible for drawing on the **results** canvas.It will step through each character in the results string and draw a black circle for a 1 and a white circle for a 0 (as shown below)



### QuestionPanel Class

| **ClassName: QuestionPanel , Inherited from Tk.Frame** |
| --- |
| **Properties**  questions: Tk.Label[] |
|  |

This will be quite a simple class whose purpose will be to display the 4 questions within the panel at evenly spaced intervals. All this can be done within the class initialisation

### ScoreBoard Class

| **ClassName: ScoreBoard , Inherited from Tk.Frame** |
| --- |
| **Properties**  current\_time: Integer  **Class References**  parent: GameScreen  **GUI Properties**  title: Tk.Label  timer: Tk.Label  score: Tk.Label  lives: Tk.Canvas  user\_details: Tk.Label |
| **Methods**  update\_scoreboard() |

#### Properties

* current\_time - this will store the time that the current level was started - it will be used to compute the number of seconds remaining to complete the level in the update\_scoreboard() method

Class References

* parent - this will be a link to the GameScreen class - this contains all the details required to be displayed on the scoreboard score, current\_level, high\_score, lives\_remaining etc..

GUI Properties

* title - used to display the world Yeardle in big font
* timer - used to display the remaining time for the current level
* score - used to display the current score
* level - used to display the current level
* lives - used to display the lives remaining
* user\_details - displays username, high\_score and high\_level for current player

#### 

#### update\_scoreboard() method

This method will be called recursively every second to update the scoreboard. It will calculate how many seconds are remaining by taking a snapshot of the current time and taking away the current\_time value stored earlier. It will check if the time remaining has been exceeded and call the loser() method of the parent GameScreen class if so. This method will also update the current score, current level and if the current score/level has exceeded the high score/level then update this as well. The method will also graphically display the number of lives remaining

## Mathematics Computations

### Projectile Motion of a Particle

If an object is projected at an angle X (to the horizontal) from a starting height of Y and at an initial velocity V under a gravitational force of G. Then we can use the quantities to find the relevant position of the object at any specific time in future T. To do this I first need to split the motion of the object into horizontal and vertical components. I can then apply the SUVAT equations to compute the displacement of the object in each of the components, which can then be used to calculate a coordinate for the current position of the object.

The SUVAT equations are listed below:

1. v = u + at
2. s = ut + 0.5at2
3. s = vt - 0.5at2
4. v2 = u2 +2as
5. s = 0.5(u+v)t

Where:

v = final velocity (m/s)

u = initial velocity (m/s)

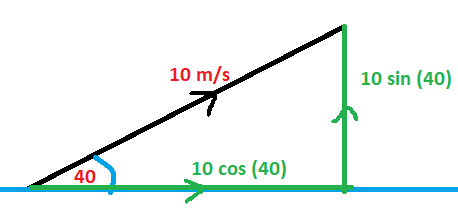
s = displacement (m)

t = time (s)

a = acceleration (m/s2)

Above you can see that apart from time (which is a scalar quantity) all the other components are vectors - this means they have both a quantity and a direction.

As mentioned earlier, before I can apply these equations, I need to split the components into horizontal and vertical. For example assume the bonus ball is to be launched at an angle of 40 degrees to the horizontal at a velocity of 10 m/s as shown below.



Using trigonometry I can calculate that the horizontal dimension of velocity can be computed as 10 cos (40) and the vertical dimension of velocity can be computed as 10 sin (40).

Now suppose I wish to compute the coordinate of a particle 2 seconds after the initial launch above. I'm going to assume the coordinate of the particle to start with was (0,0).

So to calculate position I am interested in the displacement of the particle (s). If I look at the horizontal direction I have the following known quantities and I are trying to find s:

u = 10 cos (40)

t = 2

a = 0

s = ?

Looking at the suvat equations I can see that the appropriate equation for my needs is:

s = ut + 0.5at2

Therefore:

s = 10 cos(40) \* 2 + 0.5 \* 0 \* 22

s=20 cos (40) = **15.32m**

Now looking at the vertical direction I have the following known quantities and again I am trying to calculate s (I am going to assume gravity is -9.8 m/s2, the quantity is negative because it acts downwards):

u = 10 sin (40)

t = 2

a = -9.8

s = ?

Therefore:

s = 10 sin(40) \* 2 + 0.5 \* -9.8 \* 22

s= 20 sin(40) - 19.6 = **-6.74m**

So the particle will have moved by a displacement vector of (15.32, -6.74) relative to its starting position

## Database

The database will consist of the following tables:

* Questions
  + Question (string/text, UNIQUE)
  + Answer (integer)
  + Category (string/text)
* Players
  + player \_name(string/text)
  + current\_level(integer)
  + highscore(integer)
  + highscore\_date(DATE)

A separate python file will be created to randomly generate the quiz questions from the database

# Technical Solution

## Graphical User Interface

There were 2 obvious choices for the GUI. Either a web based GUI or a windows based GUI. Given the sophisticated features such as drag/drop and bonus ball motion, it appeared that a windows based GUI would be the more appropriate solution. In order to achieve the solution as a web based GUI, it would have involved several different technologies. With a web based GUI it is impossible to build a system without extensive use of HTML, CSS and JavaScript, whereas a windows GUI can be achieved using python alone. (\*\*\*advantages of doing it on the web means you can access it on any device but this is a lot more complex to do\*\*\*)

The built-in library to support GUI development in python is called Tkinter.

Tkinter is a library where you can create an outer window and then within this place certain widgets. The main widgets I will use include the following:

* Label - this is a simple text widget
* Entry - This is a user input widget - ie this allows users to enter information into the system
* Button - This is a button that can be clicked and then linked to a specific function
* Canvas - This is quite a complicated widget that allows you draw images onto a rectangular grid

All widgets in Tkinter can be placed within the main window. This is done using an x and y coordinate system. The widgets can also be styled - eg you can change the background colour, foreground colour and font that each widget uses.

### The Main Yeardle Screen

I decided on a Pixel size of 1000 pixels wide by 500 pixels deep. To create the main area and give it a title requires the following code:

from tkinter import \*

class Yeardle(Tk):

def \_\_init\_\_(self):

Tk.\_\_init\_\_(self)

self.configure(width=1000,height=500,bg = "magenta")

yeardle= Yeardle()

yeardle.title("Yeardle!")

yeardle.mainloop()

This will create the main window where the rest of the widgets will be placed. The dimensions of the window are set using the configure method , you can also give the window a title using the title method. Note tkinter uses the Tk() class to create the main window - in this case I have simply inherited from this class.

When run this displays the following:



I now have to place all the relevant visual components - as per the design we have 8 GUI elements that must be placed on the screen. Not all of these will be visible at all times - it will depend on what stage of the game the user is at.

1. info\_screen: this will be a container for the below elements

*The info screen will be created as a Frame of 800x400 pixels with a background colour of yellow*

self.info\_screen=Frame(self,width=800,height=400,bg="yellow")

1. maintitle: this will simply display the word YEARDLE in large font

self.maintitle=Label(self.info\_screen,text=" Y E A R D L E ", font=("Arial",40),bg="black",fg="yellow")

1. message: Will display the different messages to the user at different times in the game (ie at start of game or when user loses a life or completes a level)

self.message=Label(self.info\_screen,text="message",bg="yellow")

1. stats: Will display stats for current user (eg score, level, leaderboard position)

self.stats=Label(self.info\_screen,text="stats",bg="blue",fg="yellow",font=("Arial",20))

1. playbutton: A button that will start the game when clicked

self.playbutton=Button(self.info\_screen,text="Play/Continue/Quit",bg="green")

1. addquestions: A button that will launch the UserQuestions screen to allow users to add their own questions

self.addquestions=Button(self.info\_screen,text="Add Questions",bg="orange")

1. cancel: This will be displayed with the UserQuestions screen to allow player to cancel if they have pressed the button by accident

self.cancel=Button(self,text="Cancel",command=self.display\_info,bg="red")

1. unameentry: A text entry box to allow players to enter a username

self.unameentry = Entry(self.info\_screen)

1. leaderboard: A listbox that will contain the leaderboard - a list of all players who have played before ranked by score

self.leaderboard=Listbox(self.info\_screen,width=40,font=("consolas",12),bg="blue",fg="yellow")

The above components were very basic apart from the leaderboard. In this case I used a ListBox component that is quite a bit more complicated than the Label and Button components. When adding a player to the leaderboard I needed to align the rank, name, score and level so they appeared underneath each other - this proved to be very difficult. This was to do with the fact that different names, scores etc occupied were of different lengths. To solve this I had to ‘pad’ out each of the 4 items so they all occupied the same length. However this still did not solve the problem as certain characters eg ‘w’ occupied more space on the screen that others - eg ‘l’. So in doing some research I discovered that I need to choose a font that was monospaced - this means that each character takes up the same space as all others. To this end I chose the “consolas” font and the problem was solved. The full code for displaying items in the leaderboard is as follows:

self.leaderboard.delete(0,END) #clear the leaderboard

for pos,user in enumerate(get\_leaderboard()):

rank = str(pos+1)+" "\*(3-len(str(pos+1)))

player = user[0] + " "\*(20-len(user[0]))

score = str(user[1]) + " "\*(4-len(str(user[1])))

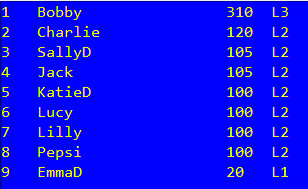
level = str(user[2]) + " "\*(4-len(str(user[2])))

value=f"{rank} {player} {score} L{level}"

self.leaderboard.insert(pos,value)

The get\_leaderboard() method will be described in more detail later but it simply returns a list that is in order of high to low score and each element of the list is a list containing the 3 items [player name, high score, high level]. I decided to allocate 3 characters to the Rank (which means a maximum of 999 rankings, 20 characters to the playername, 4 characters to the high score and 4 characters to the level. (Note this could cause some problems at a later stage if the game becomes popular). I had to then use some python string functions to pad out each value accordingly - this involved taking the value in question and converting to a string. I then added an appropriate number of spaces to the end of the string to make sure it was the required length

The code produces a Leaderboard that looks as follows:



There are 4 main screens that are displayed as part of the Yeardle Main Screen - the code and screen shots are below:

#### Welcome Screen (state = 1)

if state == 1:

if self.game:

self.game.destroy()

self.message.configure(text="Welcome to Yeardle, Please type a username below or select from the list")

self.message.pack()

self.unameentry.pack()

self.stats.forget()

self.leaderboard.delete(0,END) #clear the leaderboard

for pos,user in enumerate(get\_leaderboard()):

rank = str(pos+1)+" "\*(3-len(str(pos+1)))

player = user[0] + " "\*(20-len(user[0]))

score = str(user[1]) + " "\*(4-len(str(user[1])))

level = str(user[2]) + " "\*(4-len(str(user[2])))

value=f"{rank} {player} {score} L{level}"

self.leaderboard.insert(pos,value)

self.leaderboard.pack()

self.playbutton.configure(text="Game On", command=self.game\_on)

self.addquestions.configure(text="Add Questions", command=self.add\_questions)

self.playbutton.pack()

self.addquestions.pack()



#### Game Over Screen (state = 2)

elif state == 2:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.addquestions.forget()

msg=f"Game Over - your score was {self.game.score} and you reached Level {self.game.current\_level}"

if self.game.score > self.high\_score:

pos = self.get\_leaderboard\_position()

msg+=f"\nThis is a new high score - your position is now: {pos}"

self.message.configure(text=msg)

self.display\_stats()

self.playbutton.configure(text="OK",command=self.display\_info)



#### Life Lost Screen (state = 3)

elif state == 3:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.addquestions.forget()

self.message.configure(text=f"You have lost a life - you have {self.game.lives\_remaining} remaining")

self.leaderboard.forget()

self.unameentry.forget()

self.stats.pack()

self.playbutton.configure(text="Continue",command=self.game.start\_level)

self.display\_stats()



#### Completed Level Screen (state = 4)

elif state == 4:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.addquestions.forget()

self.message.configure(text=f"Congratulations you have completed level {self.game.current\_level-1}")

self.display\_stats()

self.leaderboard.forget()

self.unameentry.forget()

self.playbutton.configure(text="Continue",command=self.game.start\_level)



### DropZone

The DropZone will be implemented as a Python class which inherits from the Tkinter Frame class. As per the design above the GUI elements are coded as follows:

class DropZone(Frame):

def \_\_init\_\_(self,parent):

Frame.\_\_init\_\_(self,parent)

#Class References

self.parent=parent #reference to the GameScreen class

#GUI Properties

self.configure(width=80,height=300,bg="blue")

self.results= Canvas(self,width=40, height=40,bg="blue",highlightthickness=0, relief='ridge')

self.results.place(x=20,y=5)

self.done\_button = Button(self, text = "DONE", bg = "deeppink", fg = "white", font = ("Arial",14),command = self.show\_result)

self.done\_button.place(x=5,y=250)

self.done\_button["state"]="disabled"

self.targets = []

y\_coord = 50

for target in range(0,4):

lbl = Label(self, width=6,bg = "cyan")

lbl.place(x=15,y=y\_coord)

lbl.position = target

lbl.update()

self.targets.append(lbl)

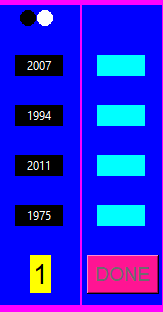
y\_coord += 50

The colours above will most likely change but they were chosen during development so I could see clearly where each component was on the screen. The DropZone frame is set to 80x300 and given a background colour of Blue.

The results area of the screen is a Canvas object which is a square 40x40 pixels with a background colour of Blue. This normally appears with a border which looked ugly so i discovered this can be removed by setting highlighthickness=0 and relief=”ridge”

The Done button is a Button object with the text “DONE” in a large Font (14) with background colour as deep pink and text colour White. The key point here is that when the user clicks the button then the method show\_results() of the DropZone class will be called.

A snapshot from the game shows the DropZones to appear as follows:



### Bonus Ball

The bonus ball will be implemented as a Python Class. The bonus ball itself will be a Tkinter Canvas widget so I investigated the best way to do this. It turns out that the most effective method is to subclass the actual Tkinter Canvas class as follows:

from tkinter import \*

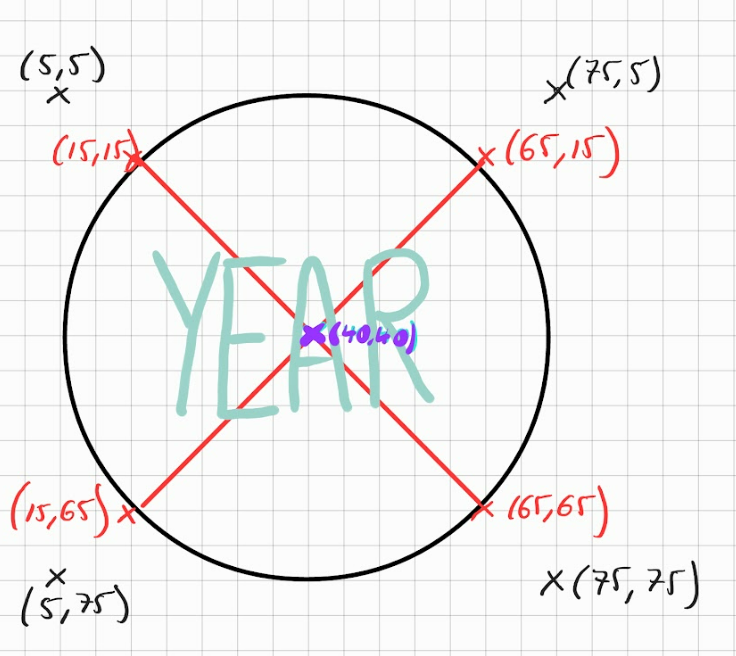
class BonusBall(Canvas):

def \_\_init\_\_(self,parent,average\_bonus\_ball\_time):

Canvas.\_\_init\_\_(self,parent)

#### Drawing the Bonus Ball on the Canvas

As mentioned previously - the Canvas is a rectangular area - I decided to make the area a square 80 pixels wide, by 80 pixels tall. In the design section above I sketched out how the bonus ball would look:



I chose to develop the image in a separate project first and then I would move the code over once I had a working prototype.

I created a new python script and generated a window to place my bonus ball in - the code was as follows:

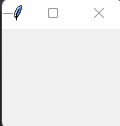
from tkinter import \*

main = Tk()

main.geometry("100x100")

main.mainloop()

When run this displays an empty window as follow:

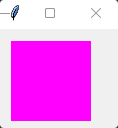


I then created an empty canvas and placed this (at coordinate (10,10) )in the window as follows:

c= Canvas(main,width=80,height=80,bg="magenta")

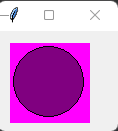
c.place(x=10,y=10)

When run this displays the following:



Now using my design above I know I needed to draw a circle with top left coordinate of (5,5) and bottom right coordinate of (75,75), I also wanted to colour the inside of the circle purple. This is done using the following code:

c.create\_oval(5,5,75,75,fill = "purple")



To create the cross I need to draw 2 diagonal lines - based on my design above the first one starts at coordinate (15,15) and ends at (65,65) and the 2nd one starts at (15,65) and ends at (65,15). Both lines were to be coloured red and the thickness of the lines was to be double the normal thickness. To do this I needed these 2 lines of code:

c.create\_line(15,15,65,65,fill = "red",width = 2)

c.create\_line(15,65,65,15,fill = "red",width = 2)

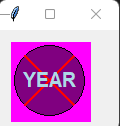
The image now looks as follows:



The final stage was to place the word YEAR in the centre of the canvas. Since the canvas has dimensions 80 by 80 then the mid point is (40,40). I also needed to make the text large enough to be easily visible - this was done by setting the font property. The line of code to achieve this was as follows:

c.create\_text(40,40,text = "YEAR",font = ("Arial",15,"bold"),fill = "lightblue")

The image is now complete and looks as follow:



The final stage was to convert all this code into a class and test that it works. This was done as follows (Note i removed the border around the canvas by setting the highlightthickness property to 0)

from tkinter import \*

class BonusBall(Canvas):

def \_\_init\_\_(self,parent):

Canvas.\_\_init\_\_(self,parent)

#create a square canvas and place the word YEAR with a red cross on it within a circle

self.configure(width=80,height=80,bg="magenta",highlightthickness=0, relief='ridge')

self.create\_oval(5,5,75,75,fill = "purple")

self.create\_text(40,40,text = "YEAR",font = ("Arial",15,"bold"),fill = "lightblue")

self.create\_line(15,15,65,65,fill = "red",width = 2)

self.create\_line(15,65,65,15,fill = "red",width = 2)

#TEST THE CODE

main = Tk()

main.geometry("300x300")

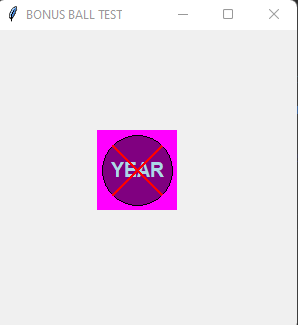
main.title("BONUS BALL TEST")

bb = BonusBall(x)

bb.place(x=100,y=100)

main.mainloop()

Running this displays the following:



### ScoreBoard

The GUI for the scoreboard is quite simple and again follows the design above. It involves placing a series of Label widgets at specific points on the screen. The only slightly complicated part is the display or remaining lives but this uses a Canvas with circles in a very similar way to the results section of the DropZone.

The code for the ScoreBoard GUI is as follows:

class ScoreBoard(Frame):

def \_\_init\_\_(self,parent):

Frame.\_\_init\_\_(self,parent)

self.parent=parent

self.configure(width=1000,height=50,bg="green")

self.title=Label(self,text="Y E A R D L E",font=("Constantia",26),bg="green",fg="yellow")

self.title.place(x=400,y=4)

self.current\_time = time.time()

self.timer=Label(self,text="TIME: ")

self.timer.place(x=5,y=5)

self.score=Label(self,text=str(self.parent.score), font=("Arial",18))

self.score.place(x=60,y=5)

self.level=Label(self,text=str(self.parent.current\_level), font=("Arial",18))

self.level.place(x=950,y=5)

self.lives=Canvas(self,width=70,height=30)

self.lives.place(x=850,y=5)

self.user\_details=Label(self,text=f"{self.parent.user}\n{self.parent.high\_score} L{self.parent.high\_level}")

self.user\_details.place(x=700,y=5)

A screenshot of how this appears when run is below:



### YearWidget

The YearWidgets are inherited from the Tkinter Label class and have a very simple GUI setup. The GUI code is as follows:

class YearWidget(Label):

POS\_COUNT = 0

COLOURS=["Yellow","Red","Green"]

def \_\_init\_\_(self,parent,year,x,y,dropzone,guesses):

Label.\_\_init\_\_(self,parent)

self.colour\_index=0

self.configure(text = str(year),bg=YearWidget.COLOURS[self.colour\_index] ,font=("Arial",10))

A user can change the colour of a YearWidget to be either Yellow, Red or Green. This is to help the player keep track of their guesses (they can mark a YearWidget as Red if they think it is a wrong answer and Green if they think it is one of the correct answers - they do this by right clicking the YearWidget with the mouse button). An example of how the YearWidgets will look within the game is as follows:



### QuestionPanel

This is the simplest class in the game and involves displaying 4 labels within a frame - the code looks as follows. The colours are likely to change but have been again chosen so I could clearly see the layout of the widgets during development. The code has a for loop that runs through each question and creates a label with the question contained within. Each Label is placed 50 pixels below the previous one.

The code is as follows

from tkinter import \*

class QuestionPanel(Frame):

def \_\_init\_\_(self,parent,questions):

Frame.\_\_init\_\_(self,parent)

self.configure(width=510,height=200,bg="orange")

y\_val = 0

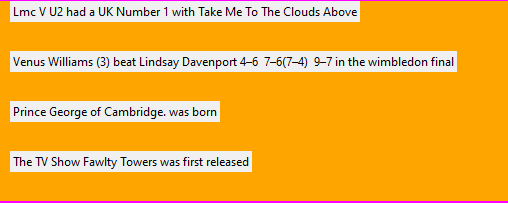
for question in questions:

lbl = Label(self,text = question,wraplength=500)

lbl.place(x=10,y=y\_val)

y\_val += 50

Below is a screenshot of how the QuestionPanel looks:



### GameScreen

This brings together many of the GUI elements covered above. Unlike other classes the GUI creation will be done outside of the class initialisation. This is because I only need to create the elements once the user has clicked the Game On button from the Yeardle class.

The code for creating the GUI elements will be contained with the start\_level() method. The relevant code is below:

self.bonus\_ball = BonusBall(self,self.bonus\_ball\_frequency)

self.bonus\_ball.years= self.years

self.bonus\_ball.answers = self.answers

self.dropzone=DropZone(self)

self.dropzones.append(self.dropzone)

self.dropzone\_pos=550

self.dropzone.place(x=self.dropzone\_pos,y=60)

self.scoreboard = ScoreBoard(self)

self.scoreboard.place(x=0,y=0)

self.questionpanel = QuestionPanel(self,self.questions)

self.questionpanel.place(x=0,y=110)

YearWidget.POS\_COUNT=0

self.year\_widgets = []

current\_y = 60

current\_x = 10

for year in self.years:

current\_year = YearWidget(self,year,current\_x,current\_y,self.dropzone,self.guesses)

current\_x += 40

self.year\_widgets.append(current\_year)

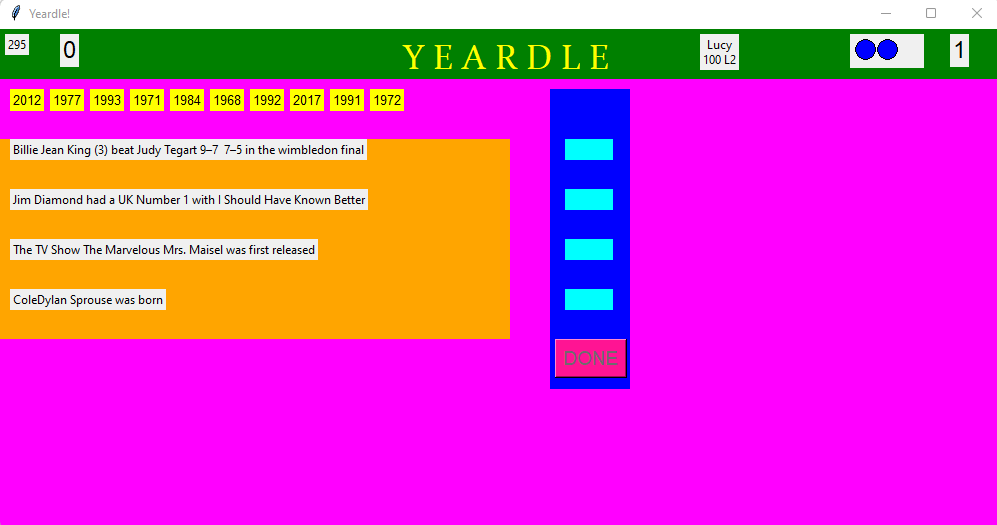
#pass a reference to years, year\_widgets and answers to bonus ball

self.bonus\_ball.year\_widgets= self.year\_widgets

In the above code I will first create an instance of the BonusBall, then I will create an instance of the DropZone and add it to the dropzones list. I will place the first DropZone at position (550,60) - each new DropZone will be placed alongside the previous one. Next I will create the ScoreBoard and place it at position (0,0). The QuestionPanel is then created and placed at position (0,110).

Finally I will create 10 instances of the YearWidget class and place them at (10,60), (50,60), (90,60) etc.. (ie all at y=60 but with an x coordinate that increases by 40 pixels each time.

When all the GUI elements have been placed on the screen it will look like the following:



### User Questions Screen

Due to the complexity of the rest of the program I decided that this was an unnecessary addition so decided to exclude it from the implementation as it would have had little impact on the rest of the program.

## Behaviour Functionality

The 8 classes are responsible for their own visual display and also for managing their own behaviour. This section covers the behaviour functionality for each class

### BonusBall

The Bonus has relatively complex functionality - there are 3 methods as follows:

#### move() method

As described in the design section this method is responsible for moving the ball across the screen in a natural way so that the player can anticipate its location and click on it.

The Ball will either be moving or waiting to move. The code for the ball moving is as follows:

self.flight\_time += 0.005

angle\_in\_radians = self.angle\_to\_horizontal \* (math.pi / 180)

horizontal\_displacement = self.start\_velocity \* math.cos(angle\_in\_radians) \* self.flight\_time

vertical\_displacement = (self.start\_velocity \* math.sin(angle\_in\_radians) \* self.flight\_time) +(0.5 \* self.vertical\_acceleration \* self.flight\_time\*\*2)

if self.start\_x == 0:

x = int(self.start\_x + horizontal\_displacement \* 200)

else:

x = int(self.start\_x - horizontal\_displacement \* 200)

y = int(self.start\_y + vertical\_displacement \* 400)

self.place(x=x,y=500 - y)# flips y coordinate as y coordinate is opposite in python

if x > 1000 or x < 0: self.reset()

The first line increases the flight\_time by a fixed amount - after experimenting I found that 0.005 seconds was a reasonable time to ensure the position changes in the SUVAT equations were not too dramatic. The Python trig functions require all angles to be converted to Radians so the 2nd line does this by dividing the angle in degrees by π/180 (as 360 degrees = 2π radians). Next we need to calculate the horizontal displacement by multiplying the initial velocity by the Cosine of the angle in radians and then by the flight\_time. The vertical displacement is computed as follows:

vertical\_displacement = ut + 0.5at2

where

u = start\_velocity \* sin(angle\_in\_radians)

t = flight\_time

a = vertical\_acceleration

In the reset() method below I set the start conditions of the Bonus Ball to random variables. One of these determines if the ball will fly from left to right (start\_x = 0) or right to left (start\_x = 1000). So in this case I need to compute the new x coordinate of the Bonus Ball accordingly (i.e. if from left to right then I will add the horizontal displacement and if from right to left we subtract it). I also need to scale the numbers so that they are in proportion to the screen - in this I computed that a scale factor of 200 for the horizontal displacement and 400 for vertical keeps the ball within the right boundaries. Once I have computed the new (x,y) coordinate I will place the Bonus ball in this new position (Note because the y coordinate runs from top to bottom I need to ‘flip’ its value by subtracting it from 500). Finally I need to check if the Bonus Ball position has gone outside the bounds of the screen - if so then the Ball has now been missed by the player and will need to be reset.

If the ball is not moving (ie bonus\_start = False) then I need to randomly decide if I need to move the ball. This will depend on the average\_bonus\_ball\_time which represents the number of seconds on average this bonus ball will be launched. It was decided that the move() method will be called every 10 milliseconds - this is done recursively as follows:

self.after(10, self.move)

So to determine if the Bonus Ball should be allowed to move on each call I need to generate a random number between 0 and 100\*average\_bonus\_ball\_time. (this is because I will be checking 100 times per second). For example if the average\_bonus\_ball\_time was 10 seconds then I would generate a random number between 0 and 1000 every 1/100th of a second. I would now compare this to a specific value (any value will do but I chose 5) and if the random number is 5 then I will allow the Ball to move. Since in the example given the random number would be generated 1000 times over a 10 second period so I would expect to get the number 5 on average once in each of these 1000 attempts. (Since the probability of any number is 1 in 1000). Finally I need to also check that maximum allowed bonus balls have not already been exceeded for the current level.

The code to handle all this is as follows:

r = random.randint(0,100\*self.average\_bonus\_ball\_time)

if r == 5 and len(self.years) >= self.parent.max\_bonus\_balls:

self.bonus\_start = True

note len(self.years) tells me how many YearWidgets are left so I compare this to the max\_bonus\_balls.

#### reset() Method

As described in the design section this method resets the ball and generations random starting conditions. It also sets some factors to their default values - namely flight\_time is set to 0 and the ball is removed from the screen. The program now computes random values as follows:

start\_y = a random value between 150 pixes and 300

start\_x = a random value of either 0 or 1000

angle\_to\_horizontal - a random value between 1 and 60 degrees

start\_velocity - a random value between 3 and 7

The final code is as follows:

def reset(self): #removes ball from screen and resets start values

self.bonus\_start = False

self.flight\_time = 0 #set the current flight time of ball to 0

self.place(x=-100,y=-100)# this makes the ball dissapear

self.start\_y = random.randint(150,300) #sets vertical launch position

if random.randint(0,1):

self.start\_x = 0 #start ball from left of screen

else:

self.start\_x = 1000 #start ball from right of screen

self.angle\_to\_horizontal = random.randint(1,60)

self.start\_velocity = random.randint(3,7)

#### scored() Method

In the initialization of the BonusBall class I set up a Binding for the left mouse button as follows:

self.bind("<Button-1>",self.scored) #bind to left mouse button

This will call the scored() method if the player clicks on the Bonus Ball. If this is the case then it means the player has successful ‘hit’ the bonus ball. In this case I need to increase the player score by 5 points and also randomly remove one of the YearWidgets. I can only remove a YearWidget that has not already been placed so the first thing to do is find out all the YearWidgets which have not been placed and are not one of the valid answers - I do this as follows:

potentials = []

for year in self.year\_widgets:

if year.year not in self.answers and year.in\_position == False:

potentials.append(year)

So I end up with all the candidates stored in the potentials list. The next thing to do is shuffle this list and take the first item. Finally I remove both the YearWidget and the Year from the game data and call the reset() method to start the process again. The final code for this method is as follows:

def scored(self,data):

potentials = []

self.parent.score+=5

for year in self.year\_widgets:

if year.year not in self.answers and year.in\_position == False:

potentials.append(year)

random.shuffle(potentials)#this randomly shuffles the years that could be removed

print (potentials[0].year,self.years)

self.years.remove(potentials[0].year)

self.year\_widgets.remove(potentials[0])

potentials[0].destroy()

self.reset()

### DropZone

As detailed in the design section, the DropZone class has a single method called show\_results()

The first thing this needs to do is call the check\_answers() method of the GameScreen class. This will return a string containing “1”s or “0”s. The approach is to create a for loop to step through each character in the string and draw a Black circle for a “1” and a White circle for a “0”. The circles should be positioned so they form 2 rows of 2 circles. Each one will be 15x15 pixels wide with a 2 pixel gap in between. This method should also remove the Done button from the screen until the next attempt and also display a number representing the attempt number in the same place where the Done button was placed.

The code is as follows (Note the create\_oval() method has been described above in the BonusBall section so is not covered again here):

def show\_result(self):

res = self.parent.check\_answers()

if res:

x = 0

y = 0

count = 1

for r in res:

if r == "1":

colour = "black"

else:

colour = "white"

self.results.create\_oval(x,y,x+15,y+15,fill = colour , outline = colour)

if count == 1 :

x += 17

elif count == 2 :

y += 17

x -= 17

elif count == 3 :

x += 17

count += 1

self.done\_button.place(x=-100,y=-100)#remove the done button

Label(self,text=str(self.parent.attempt-1),font=("Arial",20),bg="yellow").place(x=30,y=250)

The only slightly tricky bit was in the position of each of the circles. The first circle would be placed at (0,0), the second at (17,0), the third at (0,17) and the last at (17,17). I decided to keep a count of the circle and then adjust the relative (x,y) coordinates accordingly.

### GameScreen

This class has 6 methods as detailed below

#### game\_on() method

This sets the default values for a new game and the calls the start\_level() method

def game\_on(self):

self.score=0

self.lives\_remaining=3

self.current\_level=1

self.time\_allowed=300

self.bonus\_ball\_frequency=10#every 10 seconds

self.max\_bonus\_balls=6

self.start\_level()

#### start\_level() method

This calls destroy\_widgets() (see below) to reset all the GUI elements for a new level. It then places the actual game screen so it can be seen and sets up the data for this level including questions, answers etc.. It then creates a new instance of the BonusBall and add the years and answers properties to this (see BonusBall section for description of how these are used).

The first DropZone is then created and added to the list of dropzones. The ScoreBoard is created and the QuestionPanel. These 3 GUI elements are then placed at the relevant position within the screen.

Finally the YearWidgets are created in a for loop. The code for all this is shown below:

def start\_level(self):

self.destroy\_widgets()

self.place(x=0,y=0)

self.in\_play=True

self.dropzones=[]

self.attempt = 1

self.guesses = [0,0,0,0]

data = generate\_questions\_from\_db()

self.answers = data["answers"]

self.questions = data["questions"]

self.years = data["all\_years"]

self.bonus\_ball = BonusBall(self,self.bonus\_ball\_frequency)

self.bonus\_ball.years= self.years

self.bonus\_ball.answers = self.answers

self.dropzone=DropZone(self)

self.dropzones.append(self.dropzone)

self.dropzone\_pos=550

self.dropzone.place(x=self.dropzone\_pos,y=60)

self.scoreboard = ScoreBoard(self)

self.scoreboard.place(x=0,y=0)

self.questionpanel = QuestionPanel(self,self.questions)

self.questionpanel.place(x=0,y=110)

YearWidget.POS\_COUNT=0

self.year\_widgets = []

current\_y = 60

current\_x = 10

for year in self.years:

current\_year = YearWidget(self,year,current\_x,current\_y,self.dropzone,self.guesses)

current\_x += 40

self.year\_widgets.append(current\_year)

#pass a reference to years, year\_widgets and answers to bonus ball

self.bonus\_ball.year\_widgets= self.year\_widgets

#### destroy\_widgets() method

When I first developed the program I was having lots of problems with the Tkinter widgets that I was using. The program would behave in unpredictable ways as each new level progressed and I spent many hours trying to discover what was going wrong. I put lots of print statements in the program and finally discovered that the problem was that each widget was getting duplicated each time a new level was started. In order to correct this problem I had to use a special method called destroy() which would remove all previous widgets so a new set could be created. The code for this is as follows:

def destroy\_widgets(self):

if self.scoreboard:

self.scoreboard.destroy()

self.bonus\_ball.destroy()

self.questionpanel.destroy()

for yearwidget in self.year\_widgets:

yearwidget.destroy()

for dropzone in self.dropzones:

dropzone.destroy()

#### check\_answers() method

As mentioned previously when the DONE button is clicked on a DropZone instance then this method will be called (this is why the DropZone class had a property called parent so that it can communicate with the GameScreen class). This method then implements the process from the design selection and if necessary calls the loser() or winner() methods. It also creates a new DropZone for the next attempt and returns all the YearWidgets back to base. It finally increases the attempts count by 1. The code for this is as follows:

def check\_answers(self):

correct\_place = 0

incorrect\_place = 0

for i in range(0,4):

if self.guesses[i] == self.answers[i]:

correct\_place += 1

elif self.guesses[i] in self.answers:

incorrect\_place += 1

result = "1"\*correct\_place + "0" \* incorrect\_place

if result == "1111":

self.winner()

return False

elif self.attempt == 5:

self.loser()

return False

else:

#create a new dropzone

self.dropzone=DropZone(self)

self.dropzones.append(self.dropzone)

self.dropzone\_pos+=82

self.dropzone.place(x=self.dropzone\_pos,y=60)

for year\_widget in self.year\_widgets:

year\_widget.return\_to\_base()

year\_widget.lift()

year\_widget.dropzone=self.dropzone

self.attempt+=1

return result

#### winner() method

This implements the logic from the winner() design - the code is as follows:

def winner(self):

points = self.current\_level \* (6- self.attempt) \* 20

self.score+=points

self.current\_level+=1

self.time\_allowed =int(self.time\_allowed\*.9) #reduce time allowed

if self.max\_bonus\_balls > 0:

self.max\_bonus\_balls -=1

self.bonus\_ball\_frequency =int(self.bonus\_ball\_frequency\*1.1)

self.scoreboard.update\_scoreboard()

self.in\_play=False

self.parent.display\_info(4)

#### loser() method

This implements the logic from the loser() design - the code is as follows:

def loser(self):

self.in\_play=False

self.lives\_remaining -=1

if self.lives\_remaining==0:

self.parent.display\_info(2)

else:

self.parent.display\_info(3)

### ScoreBoard

This has a single method to update the scoreboard - this proved to be quite complicated as the timer needed to be updated every second and each time a player scored or lost a life etc.. it also had to be updated. I decided the easiest way was to use the same method as in the BonusBall class and use the after() method which calls itself after a preset time. I decided to update the scoreboard every 1 second (this has the slight effect of having a noticeable delay when a player scores any points). This meant I could just update all parts of the scoreboard every second which made the whole process much simpler.

To calculate the time remaining I used the Python time library which has a function called time() that will return the number of seconds that have elapsed since 1 Jan 1970. When the scoreboard is first created I take a snapshot of this time and store it in current\_time. Then each time the scoreboard is updated I take the difference between the current value and current\_time which is the time that has elapsed (in seconds).

If the current score is greater than this players highest score then I update the database with the new high score - this uses the database function update\_user\_score() described in the database section below.

In order to display the remaining lifes graphically I decided to use circles as I have in other parts of the system - this meant using the create\_oval() method as used previously.

The final code is as follows:

def update\_scoreboard(self):

elapsed = time.time() - self.current\_time

time\_remaining= self.parent.time\_allowed-int(elapsed)

if time\_remaining < 0:

self.parent.loser()

self.timer["text"]=str(time\_remaining)

self.score["text"]=str(self.parent.score)

self.level["text"]=str(self.parent.current\_level)

if self.parent.score > self.parent.high\_score:

self.parent.high\_score=self.parent.score

self.parent.high\_level = self.parent.current\_level

self.user\_details['text']=f"{self.parent.user}\n{self.parent.high\_score} L{self.parent.high\_level}"

update\_user\_score(self.parent.user,self.parent.high\_score,self.parent.high\_level)

x=5

for i in range(self.parent.lives\_remaining):

self.lives.create\_oval(x,5,x+20,25,fill="blue")

x+=22

if self.parent.in\_play: self.after(1000,self.update\_scoreboard)

### Yeardle

The main method in this class is the display\_info() method - this relies on several simpler methods. The display\_info() method checks the current state (detailed in the design section) and then implements the relevant logic. Note in all other classes I have placed all the GUI elements at specific locations using x and y coordinates. However in this case I discovered that tkinter has a method called pack() which would automatically add the GUI elements and rearrange and scale them depending on their sizes. Since this class relies on showing different information for different stages I chose this approach.

#### State 1

This represents a brand new game - this displays a welcome message and then displays an entry field for a user to enter a new user name. It also displays the leaderboard and allows the user to start the game. Note the display of the leaderboard proved to be very fiddly. I tried to use tkinter Labels at first but this proved too messy - eventually I discovered a Tkinter widget called Listbox that was much better. Each row is a string which represents multiple columns. However because the information for each column was different then the result was messy as they did not line up with each other. I decided to allocate a fixed amount of characters to each column and then pad out the rest with spaces. However this did not work with the font I was using as it turned out that all the characters were different sizes. I discovered that I could use a font where each character was the same size and decided to use the consolas font which worked perfectly but made the leaderboard look different to the rest of the game. The code for this state is as follows:

if state == 1:

if self.game:

self.game.destroy()

self.message.configure(text="Welcome to Yeardle, Please type a username below or select from the list")

self.message.pack()

self.unameentry.pack()

self.stats.forget()

self.leaderboard.delete(0,END) #clear the leaderboard

for pos,user in enumerate(get\_leaderboard()):

rank = str(pos+1)+" "\*(3-len(str(pos+1)))

player = user[0] + " "\*(20-len(user[0]))

score = str(user[1]) + " "\*(4-len(str(user[1])))

level = str(user[2]) + " "\*(4-len(str(user[2])))

value=f"{rank} {player} {score} L{level}"

self.leaderboard.insert(pos,value)

self.leaderboard.pack()

self.playbutton.configure(text="Game On", command=self.game\_on)

self.playbutton.pack()

The get\_leaderboard() method is described in the database section.

#### 

#### State 2

This is when the game is over. In this case I need to hide the game screen and display the main yeardle information screen. I expected to find a hide() method for Tkinter widgets but it did not appear to exist - the only way I could find to hide a widget was to place it outside the screen - ie using negative coordinates. In this state I will display the relevant message from the design section and change the label and behaviour of the playbutton so it now says OK and when clicked returns the game to state =1. The final code is as follows:

elif state == 2:

self.game.place(x=-1000,y=-1000) #hide the game screen

msg=f"Game Over - your score was {self.game.score} and you reached Level {self.game.current\_level}"

if self.game.score > self.high\_score:

pos = self.get\_leaderboard\_position()

msg+=f"\nThis is a new high score - your position is now: {pos}"

self.message.configure(text=msg)

self.display\_stats()

self.playbutton.configure(text="OK",command=self.display\_info)

Note this relies on a method called display\_stats(). The purpose of this is to display all relevant information about the game in progress - the code for this is as follows:

def display\_stats(self):

self.stats["text"]=f"""Score: {self.game.score}\nLevel: {self.game.current\_level}\nHigh Score: {self.game.high\_score}\nLeaderboard# {self.get\_leaderboard\_position()}"""

self.stats.pack()

#### State 3

This is shown when a user loses a life. The relevant message is displayed, the main game screen, leaderboard etc.. are hidden. The playbutton label is changed to Continue and its command is now set to start\_level. The code for this is as follows:

elif state == 3:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.message.configure(text=f"You have lost a life - you have {self.game.lives\_remaining} remaining")

self.leaderboard.forget()

self.unameentry.forget()

self.stats.pack()

self.playbutton.configure(text="Continue",command=self.game.start\_level)

self.display\_stats()

#### State 4

This is shown when the user moves to the next level. The code for this is as follow:

elif state == 4:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.message.configure(text=f"Congratulations you have completed level {self.game.current\_level-1}")

self.display\_stats()

self.leaderboard.forget()

self.unameentry.forget()

self.playbutton.configure(text="Continue",command=self.game.start\_level)

#### Other Methods

* nameselect() → this is called when a user clicks on the leaderboard - it takes the name of the user that was clicked on and sets it as the current user

def nameselect(self,data):

player=str(self.leaderboard.get(ANCHOR)).strip()

player=" ".join(player.split()).split()[1]

self.unameentry.delete(0,END)

self.unameentry.insert(0,player)

* get\_leaderboard\_position() → this uses a for loop to go through every player in the leaderboard and return the position of the current player

def get\_leaderboard\_position(self):

pos=1

for player in get\_leaderboard():

if player[0]==self.user:

return pos

pos+=1

* get\_user\_details() -> This calls the database function verify\_user() which will get the previous high\_score and high\_level for this user

def get\_user\_details(self):

details = verify\_user(self.user)

self.high\_score=details[1]

self.high\_level=details[2]

* game\_on() → this checks if the username is acceptable (i.e. between 4 and 20 characters long with no spaces. It then creates an instance of the GameScreen and calls the Gamescreen gameon() method

def game\_on(self):

self.user=self.unameentry.get()

if self.user and len(self.user) > 3 and " " not in self.user and len(self.user)<21:

self.get\_user\_details()

self.game=GameScreen(self)

self.game.game\_on()

else:

self.message.configure(text=f"Please enter a valid username")

### YearWidget

Along with the BonusBall this was the most complicated class that I had to create. A difficulty was in the drag and drop function and making this work in tkinter. I discovered that each tkinter widget could be ‘linked’ to mouse moves and button presses. I found out that each button press or movement has a special code. I also discovered that when you create a function to handle each button press some data is sent to you by tkinter. This data contains the current coordinate of the mouse and I could use this in combination with the current coordinate of the widget (using the tkinter methods winfo\_x() and winfo\_y().

After lots of experimenting I was pleased to be able to drag the YearWidgets across the screen and then when I released the left mouse button compute if and which dropzone it had landed in. To do this I had a loop that went through each of the dropzone targets. The complication is due to all coordinates being relative to the container they are in. So the x,y coordinate of the YearWidget was not using the same axis as the x,y coordinate of the dropzone target. To correct for this I had to compute the x,y coordinate of the dropzone widget itself and then add to this the x,y coordinate of the target within it. This now placed the coordinate on the same axis as the YearWidget (as they were both contained within the same ‘GameScreen’ container. I then had to compute the width and height of the target and finally call the is\_overlap method (that was detailed in the design section). If this returned True then the player had successfully placed a YearWidget on the target. In this case I needed to update the user guesses and change the target GUI so it now displayed the Year instead of an empty box.

If the YearWidget did not overlap with ANY of the targets then it was returned to base using the return to base function. The code for all of this is detailed below:

from tkinter import \*

class YearWidget(Label):

POS\_COUNT = 0

COLOURS=["Yellow","Red","Green"]

def \_\_init\_\_(self,parent,year,x,y,dropzone,guesses):

Label.\_\_init\_\_(self,parent)

self.year=year

self.dropzone= dropzone

self.guesses = guesses

self.place(x=x,y=y)

self.original\_x = x

self.original\_y = y

self.in\_position = False

self.colour\_index=0

self.configure(text = str(year),bg=YearWidget.COLOURS[self.colour\_index] ,font=("Arial",10))

self.bind("<Button-1>", self.left\_mouse\_button\_clicked)

self.bind("<B1-Motion>", self.mouse\_moving\_and\_left\_button\_clicked)

self.bind("<ButtonRelease-1>",self.left\_button\_released)

self.bind("<Button-3>",self.right\_mouse\_button\_clicked)

#event 1 - left mouse button clicked

def left\_mouse\_button\_clicked(self,data):

self.mouse\_x = data.x

self.mouse\_y = data.y

#event 2 - left mouse button held down and mouse moving

def mouse\_moving\_and\_left\_button\_clicked(self,data):

new\_x = self.winfo\_x() + data.x - self.mouse\_x

new\_y = self.winfo\_y() + data.y - self.mouse\_y

self.place(x=new\_x,y=new\_y)

#event 3 - let go of left mouse button

def left\_button\_released(self,data):

is\_target = False

x1 = self.winfo\_x() - self.mouse\_x + data.x

y1 = self.winfo\_y() - self.mouse\_y + data.y

width1 = self.winfo\_width()

height1 = self.winfo\_height()

drop\_zone\_x= self.dropzone.winfo\_x()

drop\_zone\_y= self.dropzone.winfo\_y()

for target in self.dropzone.targets:

x2 = target.winfo\_x()+drop\_zone\_x

y2 = target.winfo\_y()+drop\_zone\_y

width2 = target.winfo\_width()

height2 = target.winfo\_height()

result = self.is\_overlap(x1,y1,width1,height1,x2,y2,width2,height2)

if result == True:

if not self.in\_position: YearWidget.POS\_COUNT+=1

self.guesses[target.position] = self.year

target.configure(text=self.year,bg="black",fg="white")

is\_target = True

self.in\_position=True

self.place(x = x2, y=y2)

self["bg"] = "hotpink"

self.configure(font=("Helvetica",18,"bold"))

if not is\_target:

self.return\_to\_base() #return year widget to original position

if YearWidget.POS\_COUNT == 4:

self.dropzone.done\_button["state"] = "normal"

else:

self.dropzone.done\_button["state"] = "disabled"

def right\_mouse\_button\_clicked(self,data):

if self.colour\_index == len(YearWidget.COLOURS)-1:

self.colour\_index=0

else: self.colour\_index+=1

self.configure(bg=YearWidget.COLOURS[self.colour\_index] ,font=("Arial",10))

def is\_overlap(self,x1,y1,width1,height1,x2,y2,width2,height2):

vertex1 = [x1,y1]

vertex2 = [x1+width1,y1]

vertex3 = [x1+width1,y1+height1]

vertex4 = [x1,y1+height1]

vertices = [vertex1,vertex2,vertex3,vertex4]

for vertex in vertices:

x = vertex[0]

y = vertex[1]

if (x>x2 and x<(x2 + width2)) and (y>y2 and y<(y2 + height2)):

return True

vertex1 = [x2,y2]

vertex2 = [x2+width2,y2]

vertex3 = [x2+width2,y2+height2]

vertex4 = [x2,y2+height2]

vertices = [vertex1,vertex2,vertex3,vertex4]

for vertex in vertices:

x = vertex[0]

y = vertex[1]

if (x>x1 and x<(x1 + width1)) and (y>y1 and y<(y1 + height1)):

return True

return False

def return\_to\_base(self):

self.place(x=self.original\_x, y=self.original\_y)

self.configure(bg=YearWidget.COLOURS[self.colour\_index],font=("Arial",10))

if self.in\_position:

self.in\_position=False

YearWidget.POS\_COUNT -=1

## Database

I used the python built in database to Sqlite3

### Creating the Questions Table

First I gathered all the data via scraping the relevant web pages and storing them in a text file (see the Data Gathering section below).

I then created a table called Questions using the following code:

import sqlite3

con = sqlite3.connect("Yeardle.db")

query = "CREATE TABLE Questions(question TEXT UNIQUE, answer INT, category TEXT))"

con.execute(query)

con.commit()

The yeardlequestions.txt file (created as part of the Data Gathering process) looks as follows:

Jess Hayes & Max Morley won Love Island,2015,Celebrities

Cara De La Hoyde & Nathan Massey won Love Island,2016,Celebrities

Amber Davies & Kem Cetinay won Love Island,2017,Celebrities

Dani Dyer & Jack Fincham won Love Island,2018,Celebrities

Amber Gill & Greg O'Shea won Love Island,2019,Celebrities

Finn Tapp & Paige Turley won Love Island,2020,Celebrities

….

Each row had exactly the same format so I just had to take each line, split on the ‘,’ symbol and then create a query to add it to the database. The final code was as follows:

import sqlite3

con = sqlite3.connect("Yeardle.db")

f = open("yeardlequestions.txt")

rows = f.readlines()

for row in questions:

parts = row.split(“,”)

question = parts[0]

answer = parts[1]

category = parts[2]

query = f"INSERT INTO Questions(question,answer,category) VALUES ‘{question}’,’{answers}’,’{category}’"

con.execute(query)

con.commit()

### Creating the User Table

import sqlite3

con = sqlite3.connect("Yeardle.db")

query = "CREATE TABLE Users (username char(20) UNIQUE,highscore INTEGER, level INTEGER)"

con.execute(query)

con.commit()

### QuizQuestions.py

This was created to generate the questions to be used in each round of the game and also to create the leaderboard. It consists of the following 5 methods:

* generate\_questions\_from\_db()
* generate\_years(years)
* get\_leaderboard()
* update\_user\_score()
* verify\_user()

#### generate\_questions\_from\_db()

This method executes the following query on the database and then converts the results to a list

select \* from Questions order by random() limit 50

This query selects a random 50 questions from the database. Even though the game only requires 4 I wanted to bring back enough questions so I could filter them so they covered all categories and that no 2 years (answers) were the same.

To do this I wrote a for loop for each of the questions and then checked that neither the year nor the category had been used before. To do this I used a list to keep track of the previous Years and Categories. Once the loop had finished I took the first 4 elements of the events (question) and years (answers). The final step was to create some ‘dummy’ incorrect answers - this was done via the generate\_years() method (see below)

Finally I returned the questions, answers and dummy years as a dictionary. The code is as follows

def generate\_questions\_from\_db():

questions =list(connection.execute("select \* from Questions order by random() limit 50"))

data = []

years = []

categories = []

events=[]

for question in questions:

event = question[0]

year = question[1]

category = question[2]

if year not in years and category not in categories:

years.append(year)

categories.append(category)

#data.append(dict(event = event,year = year))

events.append(event)

years = years[:4]

data = data[:4]

all\_years = generate\_years(years)

return ({"questions": events, "all\_years": all\_years, "answers": years})

#### generate\_years()

This takes the correct 4 years and then generates a further 6 ‘dummy’ years. First I will compute the current year using the date.today().year function from the python datetime library. This is to ensure I do not create a dummy year that is in the future. Then I will take each of the correct Years and compute a random Year that is between 15 and -15 years of this. I will check that this year hasn't already been added to the dummy list and then add it. Once I have 10 years the process ends and then I use the python method shuffle to randomise the order of all the years. The code is as follows:

def generate\_years(years):

current\_year = date.today().year

all\_years = []

while len(all\_years) <= 10:

for year in years:

if year not in all\_years:

all\_years.append(year)

random\_year = year + random.randint(-15,15)

if (random\_year <= current\_year) and (random\_year not in all\_years):

all\_years.append(random\_year)

all\_years = all\_years[:10]

random.shuffle(all\_years)

return all\_years

#### get\_leaderboard()

This is a simple database query that returns all the users in the Users table ordered by high score descending. The code is as follows:

def get\_leaderboard():

query = "SELECT \* FROM Users order by highscore desc"

results= connection.execute(query)

return list(results)

#### update\_user\_score()

This runs the sql query and passes in the high\_score and high\_level as parameters. This could have been done by simply creating a string for the query using string concatenation. However whilst researching this method I discovered that this is a problem and can put a program at risk of SQL Injection. The correct way is to use parameters as the code below shows:

def update\_user\_score(username,high\_score,high\_level):

query = "UPDATE Users set highscore=?, level=? where username = ? "

connection.execute(query,(high\_score,high\_level,username))

connection.commit()

#### verify\_user()

This method first checks to see if the username currently exists in the Users table. If it does then the function will simply return the details for this user. If it does not then the new username will be inserted into the database with a high score and level of 0. This will then be returned. The code is as follows:

def verify\_user(username):

query="select \* from Users where username=?"

result = list(connection.execute(query,(username,)))

if len(result) == 0:

query = "INSERT INTO Users (username,highscore,level) values (?,?,?)"

result=[username,0,0]

connection.execute(query,(username,0,0))

connection.commit()

return result

return result[0]

## Data Gathering

I decided to gather as many questions as possible and chose the following 5 categories:

1. Music
2. Politics & History
3. Sport
4. Movies and TV
5. Celebrities

I was advised that the most efficient way to obtain relevant data was to adopt the technique of ‘Web Scraping’. This involves downloading the relevant webpage to my computer and then extracting the relevant information and then formatting it so it can be used in my program.

A web page is created using HTML - this isn't a programming language but a way of displaying information on a page. This is done using ‘tags’, for example to display text as a header for a page you might use the <h1> tag. To display information as a paragraph you would use the <p> tag and to display information as a table you would use the <Table> tag.

I decided I was going to use wikipedia as my main source of data as it is considered reliable and once I managed to scrape one page the rest would be similar. I spent some time searching for pages that contain relevant information - for example I found the following page that contained information about American Presidents:

<https://en.wikipedia.org/wiki/List_of_presidents_of_the_United_States>

I found that you could look at the HTML by right clicking in the google chrome browser and selecting the view page source. This displays the HTML that was used for the page. One option was to save this page to my computer as a file and then open it as a file in python. It would then be possible to read in each line and then use some string functions and IF statements to extract the relevant information. This would however have been very fiddly and time consuming so I investigated other solutions.

This led me to discover that a popular library called **requests** is commonly used to download webpages and load the content into a variable. However further investigation revealed this also to be quite complex.

Finally I found a solution by using a library called **pandas**. This was a perfect fit as it is used primarily for handling data in table formats such as wikipedia. In fact it even has a function for extracting tables from any webpage and storing them in a convenient format

The process for each table of data was roughly the same. I had to download the webpage into a pandas data structure (called a Data Frame) and then find which tables were relevant through a process of trial and error. I also then had to decide which of the columns were relevant (again via trial and error). Then once I extracted the data in each field I had to do further string manipulation to remove any not alphanumeric character and convert years to int etc..This was done on a case by case basis. I also had to remove any commas from the data as I was storing this in a text file where each column was separated using a comma. I discovered early on that if the data contained commas then this caused problems when I tried to use the file later.

Finally I exacted the data from the table and used it to generate questions and answers that I stored in a text file in python. The final code was as follows:

import pandas as pd

def get\_wikipedia\_tables(url,table\_index):

data=pd.read\_html(url)

return data[table\_index]

question\_number=1

questions = open("yeardlyquestions.txt","w",encoding="utf-8")

#1. Love Island Winners

url="https://en.wikipedia.org/wiki/Love\_Island\_(2015\_TV\_series)"

table\_index = 1

love\_island=get\_wikipedia\_tables(url,table\_index).values.tolist()

for person in love\_island:

location = person[3].strip()

year = int(person[6][-4:].strip())

winners = person[8].replace(","," ")

question = f"{winners} won Love Island,{year}\n"

questions.write(question)

question\_number+=1

#2. World cup final winners

url = "https://en.wikipedia.org/wiki/List\_of\_FIFA\_World\_Cup\_finals#List\_of\_finals"

table\_index=3

world\_cup = get\_wikipedia\_tables(url,table\_index)

games=world\_cup[['Year','Winners','Runners-up','Score[2]']].values.tolist()

for game in games:

try:

year=game[0]

winner=game[1].replace(","," ")

loser=game[2].replace(","," ")

score=game[3].replace(","," ")

questions.write(f"{winner} beat {loser} {score} in the world cup final,{year}\n")

except Exception as e:

pass

#3. Mens Wimbledon Final Winners

url="https://en.wikipedia.org/wiki/List\_of\_Wimbledon\_gentlemen%27s\_singles\_champions#Champions"

table\_index = 3

mens\_wimbledon = get\_wikipedia\_tables(url,table\_index)

games = mens\_wimbledon[['Year[d]','Champion','Runner-up','Score in the final[5]']].values.tolist()

for game in games:

year=game[0]

winner=game[1].replace(","," ")

loser=game[2].replace(","," ")

score = game[3].replace(","," ")

questions.write(f"{winner} beat {loser} {score} in the wimbledon final,{year}\n")

question\_number+=1

#4. Womens Wimbledon Final Winners

url="https://en.wikipedia.org/wiki/List\_of\_Wimbledon\_ladies%27\_singles\_champions"

table\_index = 3

mens\_wimbledon = get\_wikipedia\_tables(url,table\_index)

games = mens\_wimbledon[['Year[d]','Champion','Runner-up','Score in the final[5]']].values.tolist()

for game in games:

year=game[0]

winner=game[1].replace(","," ")

loser=game[2].replace(","," ")

score = game[3].replace(","," ")

questions.write(f"{winner} beat {loser} {score} in the wimbledon final,{year}\n")

question\_number+=1

#5 British Prime Ministers

url="https://en.wikipedia.org/wiki/List\_of\_prime\_ministers\_of\_the\_United\_Kingdom\_by\_length\_of\_tenure"

table\_index = 0

prime\_ministers = get\_wikipedia\_tables(url,table\_index)

people = prime\_ministers[['Prime Minister','Start']].values.tolist()

served\_already=[]

for person in people:

name = person[0].replace(","," ")

year=person[1]

if name not in served\_already:

served\_already.append(name)

questions.write(f"{name} first took office as Prime Minister of Great Britain,{year}\n")

question\_number+=1

questions.close()

## Full Code Implementation (Yeardle.py)

Bringing all of the behaviour and GUI functionality together we have the full code as follows:

from tkinter import font

import time

import random

import math

import sqlite3

from datetime import date

from tkinter import \*

connection = sqlite3.connect("Yeardle.db")

def generate\_years(years):

current\_year = date.today().year

all\_years = []

while len(all\_years) <= 10:

for year in years:

if year not in all\_years:

all\_years.append(year)

random\_year = year + random.randint(-15,15)

if (random\_year <= current\_year) and (random\_year not in all\_years):

all\_years.append(random\_year)

all\_years = all\_years[:10]

random.shuffle(all\_years)

return all\_years

def get\_leaderboard():

query = "SELECT \* FROM Users order by highscore desc"

results= connection.execute(query)

return list(results)

def update\_user\_score(username,high\_score,high\_level):

query = "UPDATE Users set highscore=?, level=? where username = ? "

connection.execute(query,(high\_score,high\_level,username))

connection.commit()

def verify\_user(username):

query="select \* from Users where username=?"

result = list(connection.execute(query,(username,)))

if len(result) == 0:

query = "INSERT INTO Users (username,highscore,level) values (?,?,?)"

result=[username,0,0]

connection.execute(query,(username,0,0))

connection.commit()

return result

return result[0]

def generate\_questions\_from\_db():

questions =list(connection.execute("select \* from Questions order by random() limit 50"))

data = []

years = []

categories = []

events=[]

for question in questions:

event = question[0]

year = question[1]

category = question[2]

if year not in years and category not in categories:

years.append(year)

categories.append(category)

#data.append(dict(event = event,year = year))

events.append(event)

years = years[:4]

data = data[:4]

all\_years = generate\_years(years)

return ({"questions": events, "all\_years": all\_years, "answers": years})

class Yeardle(Tk):

def \_\_init\_\_(self):

Tk.\_\_init\_\_(self)

self.configure(width=1000,height=500,bg = "magenta")

self.user=None

self.high\_score=None

self.high\_level=None

#Links

self.game=None #GameScreen

#GUI Widgets

self.info\_screen=Frame(self,width=800,height=400,bg="yellow")

self.maintitle=Label(self.info\_screen,text=" Y E A R D L E ", font=("Arial",40),bg="black",fg="yellow")

self.maintitle.pack()

self.message=Label(self.info\_screen,text="message",bg="yellow")

self.stats=Label(self.info\_screen,text="stats",bg="blue",fg="yellow",font=("Arial",20))

self.playbutton=Button(self.info\_screen,text="Play/Continue/Quit",bg="green")

self.cancel=Button(self,text="Cancel",command=self.display\_info,bg="red")

self.unameentry = Entry(self.info\_screen)

self.leaderboard=Listbox(self.info\_screen,width=40,font=("consolas",12),bg="blue",fg="yellow") #font needs to be monospaced

self.leaderboard.bind("<<ListboxSelect>>",self.nameselect)

self.display\_info(1)

def nameselect(self,data):

player=str(self.leaderboard.get(ANCHOR)).strip()

player=" ".join(player.split()).split()[1]

self.unameentry.delete(0,END)

self.unameentry.insert(0,player)

def get\_user\_details(self):

details = verify\_user(self.user)

self.high\_score=details[1]

self.high\_level=details[2]

def get\_leaderboard\_position(self):

pos=1

for player in get\_leaderboard():

if player[0]==self.user:

return pos

pos+=1

def display\_stats(self):

self.stats["text"]=f"""Score: {self.game.score}\nLevel: {self.game.current\_level}\nHigh Score: {self.game.high\_score}\nLeaderboard# {self.get\_leaderboard\_position()}"""

self.stats.pack()

def display\_info(self,state=1):

#state codes

#1. This is a brand new game so ask for username, display the leaderboard, and have quit of game on button

#2. It is game over from an existing game

#3. The user has lost a life from an existing game

#4. The user has proceeded to the next level

self.info\_screen.place(x=320,y=50)

if state == 1:

if self.game:

self.game.destroy()

self.message.configure(text="Welcome to Yeardle, Please type a username below or select from the list")

self.message.pack()

self.unameentry.pack()

self.stats.forget()

self.leaderboard.delete(0,END) #clear the leaderboard

for pos,user in enumerate(get\_leaderboard()):

rank = str(pos+1)+" "\*(3-len(str(pos+1)))

player = user[0] + " "\*(20-len(user[0]))

score = str(user[1]) + " "\*(4-len(str(user[1])))

level = str(user[2]) + " "\*(4-len(str(user[2])))

value=f"{rank} {player} {score} L{level}"

self.leaderboard.insert(pos,value)

self.leaderboard.pack()

self.playbutton.configure(text="Game On", command=self.game\_on)

self.playbutton.pack()

elif state == 2:

self.game.place(x=-1000,y=-1000) #hide the game screen

msg=f"Game Over - your score was {self.game.score} and you reached Level {self.game.current\_level}"

if self.game.score > self.high\_score:

pos = self.get\_leaderboard\_position()

msg+=f"\nThis is a new high score - your position is now: {pos}"

self.message.configure(text=msg)

self.display\_stats()

self.playbutton.configure(text="OK",command=self.display\_info)

elif state == 3:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.message.configure(text=f"You have lost a life - you have {self.game.lives\_remaining} remaining")

self.leaderboard.forget()

self.unameentry.forget()

self.stats.pack()

self.playbutton.configure(text="Continue",command=self.game.start\_level)

self.display\_stats()

elif state == 4:

self.game.place(x=-1000,y=-1000) #hide the game screen

self.message.configure(text=f"Congratulations you have completed level {self.game.current\_level-1}")

self.display\_stats()

self.leaderboard.forget()

self.unameentry.forget()

self.playbutton.configure(text="Continue",command=self.game.start\_level)

def game\_on(self):

self.user=self.unameentry.get()

if self.user and len(self.user) > 3 and " " not in self.user and len(self.user)<21:

self.get\_user\_details()

self.game=GameScreen(self)

self.game.game\_on()

else:

self.message.configure(text=f"Please enter a valid username")

class BonusBall(Canvas):

def \_\_init\_\_(self,parent,average\_bonus\_ball\_time,years,answers):

Canvas.\_\_init\_\_(self,parent)

self.parent=parent

self.years= years

self.answers = answers

#create a square canvas and place the word YEAR with a red cross on it within a circle

self.configure(width=80,height=80,bg="magenta",highlightthickness=0, relief='ridge')

self.create\_oval(5,5,75,75,fill = "purple")

self.create\_text(40,40,text = "YEAR",font = ("Arial",15,"bold"),fill = "lightblue")

self.create\_line(15,15,65,65,fill = "red",width = 2)

self.create\_line(15,65,65,15,fill = "red",width = 2)

self.bind("<Button-1>",self.scored) #bind to left mouse button

self.average\_bonus\_ball\_time = average\_bonus\_ball\_time

self.vertical\_acceleration = -9.8 #set vertical acceleration to gravity approximation

self.reset()

self.move()

def reset(self): #removes ball from screen and resets start values

self.bonus\_start = False

self.flight\_time = 0 #set the current flight time of ball to 0

self.place(x=-100,y=-100)# this makes the ball disappear

self.start\_y = random.randint(150,300) #sets vertical launch position

if random.randint(0,1):

self.start\_x = 0 #start ball from left of screen

else:

self.start\_x = 1000 #start ball from right of screen

self.angle\_to\_horizontal = random.randint(1,60)

self.start\_velocity = random.randint(3,7)

def move(self):

if self.bonus\_start == True:

self.flight\_time += 0.005

angle\_in\_radians = self.angle\_to\_horizontal \* (math.pi / 180)

horizontal\_displacement = self.start\_velocity \* math.cos(angle\_in\_radians) \* self.flight\_time

vertical\_displacement = (self.start\_velocity \* math.sin(angle\_in\_radians) \* self.flight\_time) +(0.5 \* self.vertical\_acceleration \* self.flight\_time\*\*2)

if self.start\_x == 0:

x = int(self.start\_x + horizontal\_displacement \* 200)

else:

x = int(self.start\_x - horizontal\_displacement \* 200)

y = int(self.start\_y + vertical\_displacement \* 400)

self.place(x=x,y=500 - y)# flips y coordinate as screen runs top to bottom

if x > 1000 or x < 0: self.reset() #ball has gone outside bounds of screen

else:

r = random.randint(0,100\*self.average\_bonus\_ball\_time)

if r == 5 and len(self.years) >= self.parent.max\_bonus\_balls:

self.bonus\_start = True

self.after(10, self.move)

def scored(self,data):

potentials = []

self.parent.score+=5

for year in self.year\_widgets:

if year.year not in self.answers and year.in\_position == False:

potentials.append(year)

random.shuffle(potentials)#this randomly shuffles the years that could be removed

print (potentials[0].year,self.years)

self.years.remove(potentials[0].year)

self.year\_widgets.remove(potentials[0])

potentials[0].destroy()

self.reset()

class DropZone(Frame):

def \_\_init\_\_(self,parent):

Frame.\_\_init\_\_(self,parent)

self.parent=parent #reference to the GameScreen class

#GUI Properties

self.configure(width=80,height=300,bg="blue")

self.results= Canvas(self,width=40, height=40,bg="blue",highlightthickness=0, relief='ridge')

self.results.place(x=20,y=5)

self.done\_button = Button(self, text = "DONE", bg = "deeppink", fg = "white", font = ("Arial",14),command = self.show\_result)

self.done\_button.place(x=5,y=250)

self.done\_button["state"]="disabled"

self.targets = []

y\_coord = 50

for target in range(0,4):

lbl = Label(self, width=6,bg = "cyan")

lbl.place(x=15,y=y\_coord)

lbl.position = target

lbl.update()

self.targets.append(lbl)

y\_coord += 50

def show\_result(self):

res = self.parent.check\_answers()

if res:

x = 0

y = 0

count = 1

for r in res:

if r == "1":

colour = "black"

else:

colour = "white"

self.results.create\_oval(x,y,x+15,y+15,fill = colour , outline = colour)

if count == 1 :

x += 17

elif count == 2 :

y += 17

x -= 17

elif count == 3 :

x += 17

count += 1

self.done\_button.place(x=-100,y=-100)#remove the done button

Label(self,text=str(self.parent.attempt-1),font=("Arial",20),bg="yellow").place(x=30,y=250)

class YearWidget(Label):

POS\_COUNT = 0

COLOURS=["Yellow","Red","Green"]

def \_\_init\_\_(self,parent,year,x,y,dropzone,guesses):

Label.\_\_init\_\_(self,parent)

self.year=year

self.dropzone= dropzone

self.guesses = guesses

self.place(x=x,y=y)

self.original\_x = x

self.original\_y = y

self.in\_position = False

self.colour\_index=0

self.configure(text = str(year),bg=YearWidget.COLOURS[self.colour\_index] ,font=("Arial",10))

self.bind("<Button-1>", self.left\_mouse\_button\_clicked)

self.bind("<B1-Motion>", self.mouse\_moving\_and\_left\_button\_clicked)

self.bind("<ButtonRelease-1>",self.left\_button\_released)

self.bind("<Button-3>",self.right\_mouse\_button\_clicked)

#event 1 - left mouse button clicked

def left\_mouse\_button\_clicked(self,data):

self.mouse\_x = data.x

self.mouse\_y = data.y

#event 2 - left mouse button held down and mouse moving

def mouse\_moving\_and\_left\_button\_clicked(self,data):

new\_x = self.winfo\_x() + data.x - self.mouse\_x

new\_y = self.winfo\_y() + data.y - self.mouse\_y

self.place(x=new\_x,y=new\_y)

#event 3 - let go of left mouse button

def left\_button\_released(self,data):

is\_target = False

x1 = self.winfo\_x() - self.mouse\_x + data.x

y1 = self.winfo\_y() - self.mouse\_y + data.y

width1 = self.winfo\_width()

height1 = self.winfo\_height()

drop\_zone\_x= self.dropzone.winfo\_x()

drop\_zone\_y= self.dropzone.winfo\_y()

for target in self.dropzone.targets:

x2 = target.winfo\_x()+drop\_zone\_x

y2 = target.winfo\_y()+drop\_zone\_y

width2 = target.winfo\_width()

height2 = target.winfo\_height()

result = self.is\_overlap(x1,y1,width1,height1,x2,y2,width2,height2)

if result == True:

if not self.in\_position: YearWidget.POS\_COUNT+=1

self.guesses[target.position] = self.year

target.configure(text=self.year,bg="black",fg="white")

is\_target = True

self.in\_position=True

self.place(x = x2, y=y2)

self["bg"] = "hotpink"

self.configure(font=("Helvetica",18,"bold"))

if not is\_target:

self.return\_to\_base() #return year widget to original position

if YearWidget.POS\_COUNT == 4:

self.dropzone.done\_button["state"] = "normal"

else:

self.dropzone.done\_button["state"] = "disabled"

def right\_mouse\_button\_clicked(self,data):

if self.colour\_index == len(YearWidget.COLOURS)-1:

self.colour\_index=0

else: self.colour\_index+=1

self.configure(bg=YearWidget.COLOURS[self.colour\_index] ,font=("Arial",10))

def is\_overlap(self,x1,y1,width1,height1,x2,y2,width2,height2):

vertex1 = [x1,y1]

vertex2 = [x1+width1,y1]

vertex3 = [x1+width1,y1+height1]

vertex4 = [x1,y1+height1]

vertices = [vertex1,vertex2,vertex3,vertex4]

for vertex in vertices:

x = vertex[0]

y = vertex[1]

if (x>x2 and x<(x2 + width2)) and (y>y2 and y<(y2 + height2)):

return True

vertex1 = [x2,y2]

vertex2 = [x2+width2,y2]

vertex3 = [x2+width2,y2+height2]

vertex4 = [x2,y2+height2]

vertices = [vertex1,vertex2,vertex3,vertex4]

for vertex in vertices:

x = vertex[0]

y = vertex[1]

if (x>x1 and x<(x1 + width1)) and (y>y1 and y<(y1 + height1)):

return True

return False

def return\_to\_base(self):

self.place(x=self.original\_x, y=self.original\_y)

self.configure(bg=YearWidget.COLOURS[self.colour\_index],font=("Arial",10))

if self.in\_position:

self.in\_position=False

YearWidget.POS\_COUNT -=1

class GameScreen(Frame):

def \_\_init\_\_(self,yeardle):

Frame.\_\_init\_\_(self,yeardle)

self.configure(width=1000,height=600)

self.in\_play=False

self.yeardle=yeardle

self.configure(bg = "magenta")

self.user=yeardle.user

self.high\_score=yeardle.high\_score

self.high\_level=yeardle.high\_level

self.scoreboard = None

def destroy\_widgets(self):

if self.scoreboard:

self.scoreboard.destroy()

self.bonus\_ball.destroy()

self.questionpanel.destroy()

for yearwidget in self.year\_widgets:

yearwidget.destroy()

for dropzone in self.dropzones:

dropzone.destroy()

def game\_on(self):

self.score=0

self.lives\_remaining=3

self.current\_level=1

self.time\_allowed=300

self.bonus\_ball\_frequency=10#every 10 seconds

self.max\_bonus\_balls=6

self.start\_level()

def start\_level(self):

self.destroy\_widgets()

self.place(x=0,y=0)

self.in\_play=True

self.dropzones=[]

self.attempt = 1

self.guesses = [0,0,0,0]

data = generate\_questions\_from\_db()

self.answers = data["answers"]

print(self.answers)

self.questions = data["questions"]

self.years = data["all\_years"]

self.bonus\_ball = BonusBall(self,self.bonus\_ball\_frequency,self.years,self.answers)

self.dropzone=DropZone(self)

self.dropzones.append(self.dropzone)

self.dropzone\_pos=550

self.dropzone.place(x=self.dropzone\_pos,y=60)

self.scoreboard = ScoreBoard(self)

self.scoreboard.place(x=0,y=0)

self.questionpanel = QuestionPanel(self,self.questions)

self.questionpanel.place(x=0,y=110)

YearWidget.POS\_COUNT=0

self.year\_widgets = []

current\_y = 60

current\_x = 10

for year in self.years:

current\_year = YearWidget(self,year,current\_x,current\_y,self.dropzone,self.guesses)

current\_x += 40

self.year\_widgets.append(current\_year)

#pass a reference to years, year\_widgets and answers to bonus ball

self.bonus\_ball.year\_widgets= self.year\_widgets

def winner(self):

points = self.current\_level \* (6- self.attempt) \* 20

self.score+=points

self.current\_level+=1

self.time\_allowed =int(self.time\_allowed\*.9) #reduce time allowed

if self.max\_bonus\_balls > 0:

self.max\_bonus\_balls -=1

self.bonus\_ball\_frequency =int(self.bonus\_ball\_frequency\*1.1)

self.scoreboard.update\_scoreboard()

self.in\_play=False

self.yeardle.display\_info(4)

def loser(self):

self.in\_play=False

self.lives\_remaining -=1

if self.lives\_remaining==0:

self.yeardle.display\_info(2)

else:

self.yeardle.display\_info(3)

def check\_answers(self):

correct\_place = 0

incorrect\_place = 0

for i in range(0,4):

if self.guesses[i] == self.answers[i]:

correct\_place += 1

elif self.guesses[i] in self.answers:

incorrect\_place += 1

result = "1"\*correct\_place + "0" \* incorrect\_place

if result == "1111":

self.winner()

return False

elif self.attempt == 5:

self.loser()

return False

else:

#create a new dropzone

self.dropzone=DropZone(self)

self.dropzones.append(self.dropzone)

self.dropzone\_pos+=82

self.dropzone.place(x=self.dropzone\_pos,y=60)

for year\_widget in self.year\_widgets:

year\_widget.return\_to\_base()

year\_widget.lift()

year\_widget.dropzone=self.dropzone

self.attempt+=1

return result

class QuestionPanel(Frame):

def \_\_init\_\_(self,parent,questions):

Frame.\_\_init\_\_(self,parent)

self.configure(width=510,height=200,bg="magenta")

y\_val = 0

for question in questions:

lbl = Label(self,text = question,font=("Arial",10),wraplength=500,bg="blue",fg="white")

lbl.place(x=10,y=y\_val)

y\_val += 50

class ScoreBoard(Frame):

def \_\_init\_\_(self,parent):

Frame.\_\_init\_\_(self,parent)

self.parent=parent

self.configure(width=1000,height=50,bg="green")

self.title=Label(self,text="Y E A R D L E",font=("Constantia",26),bg="green",fg="yellow")

self.title.place(x=400,y=4)

self.current\_time = time.time()

self.timer=Label(self,text="TIME: ",font=("Arial",18),bg="red",fg="white")

self.timer.place(x=5,y=5)

self.score=Label(self,text=str(self.parent.score), bg="green",fg="white",font=("Arial",18))

self.score.place(x=800,y=5)

self.level=Label(self,text=str(self.parent.current\_level), font=("Arial",18),bg="green",fg="white")

self.level.place(x=950,y=5)

self.lives=Canvas(self,width=70,height=30,bg="green")

self.lives.place(x=700,y=5)

self.user\_details=Label(self,text=f"{self.parent.user}: {self.parent.high\_score} L{self.parent.high\_level}",font=("Arial",16),bg="yellow",fg="green")

self.user\_details.place(x=150,y=7)

self.update\_scoreboard()

def update\_scoreboard(self):

elapsed = time.time() - self.current\_time

time\_remaining= self.parent.time\_allowed-int(elapsed)

if time\_remaining < 0:

self.parent.loser()

self.timer["text"]=str(time\_remaining)

self.score["text"]="SCORE: "+str(self.parent.score)

self.level["text"]="L"+str(self.parent.current\_level)

if self.parent.score > self.parent.high\_score:

self.parent.high\_score=self.parent.score

self.parent.high\_level = self.parent.current\_level

self.user\_details['text']=f"{self.parent.user}\n{self.parent.high\_score} L{self.parent.high\_level}"

update\_user\_score(self.parent.user,self.parent.high\_score,self.parent.high\_level)

x=5

for i in range(self.parent.lives\_remaining):

self.lives.create\_oval(x,5,x+20,25,fill="blue")

x+=22

if self.parent.in\_play: self.after(1000,self.update\_scoreboard)

#STARTS THE GAME!

yeardle= Yeardle()

yeardle.geometry("1000x500")

yeardle.title("Yeardle!")

yeardle.mainloop()

# Testing

## Testing Structure

The game was built as 5 major versions:

1. Non Graphical Text Version (No Database)
2. Non Graphical Text Version (With Database)
3. Graphical Non Object Oriented Version
4. Graphical OOP Version (unstyled)
5. Graphical OOP Version Fully Styled

At each stage much of the code was tested and then reused in subsequent stages - this formed the Iterative Testing stage. The below is not a comprehensive list of all the testing in the program as this would run to many pages - it does however cover the main tests and the main issues I encountered during the development of the program.

## Iterative Testing

### Non Graphical Text Version (No Database)

Originally the game was built as a text based game - this allowed me to focus on some of the key game features and test that they worked before moving on - the features and tests carried out at this stage were:

| **Feature: Produce Question List** |
| --- |
| Approach: I Hardcoded 20 questions into a python array and then created functions to generate 4 questions and answers with a set of ‘dummy’ wrong answers |
| Tests:   | **Test** | **Result** | | --- | --- | | Does the function generate 4 unique questions from different categories? | I had problems with the categories being duplicated at first so I added a list to track which categories had already been used and then the test passed | | Does the function generate 6 ‘dummy’ answers | Several problems arose here - sometimes the program added years in the future, duplicate years and years that were the same as the actual correct answers. I solved these by finding the current year and ensuring no year was added that was beyond this. I also added a comparison between the correct year and the ‘random’ year and didnt add if they were the same. I also ensured that if a year was in the list of already added years then it would not be added. The program had to be adapted to continue attempting to add dummy years until all these conditions were satisfied | |  |  | |

| **Feature: Question Web Scraping** |
| --- |
| Approach: I used a single web page and the pandas library to create a text file containing the questions and answers. The web page i used was from wikipedia https://en.wikipedia.org/wiki/List\_of\_FIFA\_World\_Cup\_finals#List\_of\_finals |
| Tests:   | **Test** | **Result** | | --- | --- | | Does the program correctly download the data | This was straightforward as the pandas function read\_html(url) successfully downloaded the data | | Did the program manage to convert the data into a usable python list? | Firstly I had problems here as the pandas function downloaded the data into a structure called a DataFrame and there were many tables in this structure. This required me to use trial and error to find which one of the tables was relevant. Then I couldn't use any of the data without learning about DataFrames which was beyond the scope of my project. Eventually I found a way of converting the table into a regular python list and the test was successful | | Did the program convert the list into a set of useable question and answers | The list created was actually a list of lists - so I had to use trial and error to figure out which fields represented the question and which represented the answer. A bigger issue was that some items had fewer elements than others so for example the score of the game was usually at element 3 but sometimes it was at element 2. This meant that when my program tried to access element 3 it crashed. To save time I decided to put a try catch exception block around this loop to ignore any non standard items. This proved much quicker than trying to solve each problem on an individual basis and I still had plenty of questions that could be used | | Did the program successfully store the questions and answers to a file | There was one big problem with this. I had decided to use a comma to separate the question,answer and category. However some of the questions had a comma already in them so the question files had extra columns on some rows. I solved this by using the python replace() function to remove any commas from the data and the test succeeded | |
| Comments: Each of the many web pages I used had a set of small unique problems that required iterative testing. This was mainly due to the data having a structure that was not expected so I had to use string manipulation to ‘clean’ the data into a format that could be used |

| **Feature: Mastermind Game Play** |
| --- |
| Approach: This involved presenting 4 questions to the user and displaying the 10 possible answers then computing the result of each of their guesses |
| Tests:   | **Test** | **Result** | | --- | --- | | Does the function display the correct questions and choices list? | The question part was no problem but the years were displayed with the correct 4 answers first and the 6 incorrect after. This required me to go back to an earlier feature and change the dummy years so they are shuffled in a random order. This is something that was not picked up in that original test | | Does the function display the correct response to a users ‘guess’ | I had to compare each of the guesses to the correct answers and then produce a response of “1” or “0” or nothing. At first these were added in order of the questions but then it became obvious to the player which ones were in the wrong place and which were in the correct. To resolve this I changed the answers so all the 1s were displayed first and the 0s were displayed after. This made the test successful | |  |  | |

### Non Graphical Text Version (With Database)

The next stage was to move all the text file data to a database and then replace the question generation functions with new functions. This was much less work than it would have been had the previous testing not been carried out. I now had all the data i required in the correct format and also the logic for generating questions and answers

| **Feature: Convert previous version to use Database** |
| --- |
| **Approach:** I wrote a program that opened the file that i had created previously, read in each line and then added that to the database. |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Do the question in the database match the ones in the text file | I had set my database up to have the question field as a unique field. When i ran my program I discovered several questions were duplicated and the program threw errors when trying to add new data. There must have been some problem in the previous stage when adding the data to the text file but I hadn't checked for duplicates so it wasn't picked up. Again i used a try catch block which meant that duplicates were not added and the test passed | | Does the new question generation feature work the same as the original? | The main problem here was the database returning questions in the same order every time so the question list was the same. I solved this by changing the query to select \* from Questions order by random() | |

### Graphical Non Object Oriented Version

Now the basic game was working with a database I needed to add in the graphical features. This was easily the most complicated part of the whole project and required extensive testing.

| **Feature: Create a flying bonus ball** |
| --- |
| **Approach:**I created an empty Tkinter window with a single Label widget and then created a function to have this widget fly across the screen. Originally I hardcoded all the ‘flight’ values so the tests were not dealing with random values each time |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Does the ‘Label’ successfully fly from one side of the screen to the other | The maths part wasn't too tricky as I just needed to implement SUVAT functions to calculate the position of the widget at a specified time. The 2 biggest issues were the units and the display of the widget on the screen. At first I used a loop to represent seconds but that ran so fast the widget disappeared as soon as the program was run. I then introduced a ‘pause’ into the loop which solved that problem but the widget still flew straight off the screen in large steps. I had to use trial and error to play with the time value and also the displacement values so they matched the dimensions of the screen and the flight was at a suitable speed. | |  |  | |

| **Feature: Making the bonus ball fly randomly** |
| --- |
| **Approach:** The bonus ball now had to fly randomly from left to right, at different speeds and at different angles. I created variables for all these factors and then set them to random variables before each ‘flight’ |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Does the ‘Label’ successfully fly randomly | The biggest issue here was choosing suitable ranges for the random values as if they were too big or small the widget would fly too fast, too slow or at an angle that was too high. Again these were all solved via trial and error | |

| **Feature: Creating Drag and Drop functionality** |
| --- |
| **Approach:** I created a single Tkinter Label widget and attempted to pick it up and drop it in a different location |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Does the ‘Label’ successfully drag and drop | I had many issues with this - at first the label travelled in the opposite direction to my mouse pointer so I had to ‘flip’ the coordinates as I had not taken into account that the screen coordinates go from the top left hand corner. Also I didn't take into account the initial start position of the widget as I hadnt added a function for when the left mouse button is initially clicked. These issues were again fixed by trial and error and repeat testing | |

| **Feature: Dropping a YearWidget onto a DropZone** |
| --- |
| **Approach:** I created a single label widget that represented a dropzone target and used the label widget from the previous test to represent the year widget |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Does the program detect a successful drop? | This required the creation of a complex function to compute the overlap of 2 rectangles. Getting this correct involved a lot of trial and error. I tried to use the width and top left hand corner of each rectangle but realised a better method would be to compute the 4 vertex of each rectangle and work out if a single vertex from the first rectangle lay within the bounds of the second. This proved successful | | Does the program handle a successful drop | One a successful drop has been detected the system had to place the YearWidget onto the dropzone. I did this by placing both widgets at the same location and this worked | | Does the program handle a failed drop | The issue here was that if the drop was unsuccessful the program simply displayed the YearWidget outside of the dropzone. I realised I need to move the widget back to its original starting location to prevent this problem so I introduced a function called ReturnToBase() to handle this. | |

### 

| **Feature: Designing and Placing all the widgets on screen** |
| --- |
| **Approach:** I created a widget for all the items in the game - BonusBall, DropZone, DropZone Target, Question Panel, Scoreboard etc… I then placed them all at the predesigned locations on the screen |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Did all the widgets get successfully designed and placed | This was a very time consuming effort involving lots of trial and error. At this stage I chose all the colours to be bright and different to each other so I could identify the different widgets on the screen during the testing phase. This phase continued throughout the development of the game as I had to make constant small adjustments to the size and position of elements | |

| **Feature: Combining all the UI Elements** |
| --- |
| **Approach:**This involved testing the elements and features all worked together as expected |
| **Tests:**   | **Test** | **Result** | | --- | --- | | Did all the program run as expected | The biggest issue here was to do with the bonus ball. When it flew across the screen the rest of the game froze so a player couldn't click on it or move the Year Widgets. I realised that the pause feature I had introduced to slow the ball down was creating the problem as it paused the entire program. To fix this I had to use a Tkinter function called after() - this is a function that will call another function after a specified amount of time. I realised If i removed the loop and instead used a function that would call itself recursively every 1/100th of a second that the problem would be fixed. After this the testing mostly involved extensive trial and error to allow the widgets to work together | |

### 

### Graphical OOP Version (unstyled)

At this stage I had a fully working program - however this was not using objects - it was simply a bunch of functions in a single python program. This was done as a way of testing each item iteratively. Converting the code to objects proved much easier at this stage. I created a class for each component and then moved the code related to that component into the class and converted the functions to methods. After each component was complete I tested the whole program to see if it still worked. The only major issue that arose at this point was due to my use of Global Variables in the non OOP version. These were no longer visible to the individual classes. In order to solve this I had to pass each global variable as a parameter to the init method of the class and then store this as a member variable. As this process evolved and I converted each class I realised that in many cases I could simply pass the object instance of the class that contained these variables instead of each individual variable.

### Graphical OOP Version Fully Styled

The final stage was to style the individual components by changing their colours. This was simply to make the program appear more attractive and did not involve any significant testing.

## Post Development Testing

The program had mainly been tested using iterative techniques but I required a full system test to ensure the entire program worked as planned. The entire test script that I followed is laid out below - there are 21 steps to check that all the key features of the program are working:

A full run through of this test script can be seen here: <https://youtu.be/iEXfgVlUQPI>

| **#** | **Test** | **Expected Response** |
| --- | --- | --- |
| 1 | Run the program | Program screen appears showing leaderboard and Game On button |
| 2 | Click on individual users and check their name appears in the Entry Box | Name selected highlights and displays in entry box |
| 3 | Type a name that is too short or too long (<4 or >15 characters) or has spaces then click Game On | System asks you to enter a valid username |
| 4 | Click a name and select Game On | The game screen appears as follows |
| 5 | On the Game Screen check that all the widgets are displayed correctly | The following elements should be visible:   1. Countdown timer (should be counting down) 2. Current High Score/Level for Player 3. The Number of Lives remaining 4. The Current Score 5. The Current Level 6. A list of 10 Year Widgets 7. The Question Panel displaying 4 Questions 8. A Single Drop Zone featuring 4 Target Areas and a Done button. The Done button should be disabled at this stage |
| 6 | Right Click a Year Widget Once | It turns Red |
| 7 | Right Click Same Year Widget again | It turns Green |
| 8 | Right click it a 3rd time | It turns back to Yellow |
| 9 | Drag a widget and drop it on a target DropZone | It becomes attached to the dropzone and is removed from the YearWidget ‘base’ area |
| 10 | Drag a widget and drop it outside a target DropZone | The widget returns back to base |
| 11 | Drag 4 Year Widgets to the target areas | The Done button should now become enabled |
| 12 | Click the Done Button | The result of the guess should be displayed (in this case we see a Black and a White counter). The colour of the background and foreground of the guesses change to show that is now an old guess. We also see that this was attempt number 2 and finally a new dropzone should be created alongside this. |
| 14 | Place 4 correct answers in the DropZone and click the DONE button | In test mode the answers are printed out to the terminal  Using these answers should result in the following screen appearing |
| 15 | Click Continue | A new game screen should appear and the level should have increased by 1 |
| 16 | Click on the Bonus Ball | Your score should be increased by 5 points and one of the (incorrect) YearWidgets should be removed from the screen |
| 17 | Allow the time to run down | Screen appears stating you have lost a life and displays your current game statistics |
| 18 | Click Continue after losing a life | The game screen appears again with a new set of questions and the scoreboard should show 1 less life remaining |
| 19 | Do 5 incorrect guesses and then click DONE | Again you should be shown the screen stating you have lost a life |
| 20 | Lose all 3 lives | Screen appears stating that its game over. |
| 21 | Click OK | The Game Should return to the first screen and the whole process can now be repeated |

# Evaluation

Overall I was very pleased with the result and believe that I successfully met the original objectives. Below I evaluate each objective based on my own opinion and that of others who have played the game

## Educational Experience Through General Knowledge

The specific objective was met based on the target of having events covering a wide range of topics - I successfully added questions in each of the key topics of Sports, Music, Politics & History, Movies and Celebrities. The educational experience was however quite limited given that it only referred to the year that an event happened. When friends and family played the game they largely enjoyed the questions especially when they already knew the answers and some questions led to lively debates which was fun to watch. However many questions were considered too hard and were sometimes off putting. In particular there were a lot of football questions in the game that seemed to dominate the sports category so that other sports such as tennis and athletics rarely came up. In future it would be beneficial to have a more even balance of questions presented either by changing the original list of questions or changing the code so that it selects a wider variety for each question set. Another way to solve this problem would be to allow user questions to be added to the database - this feature was actually included in the original scope but had to be removed as the program proved to be much larger than I originally thought.

Another criticism about the questions was that they did not progress in difficulty as the levels increased. Players would have enjoyed easier questions in the early rounds that get more difficult with each level. This again was beyond the scope of my project as I would have had to categorise each question manually which would have been fiddly and time consuming. A future version could introduce some artificial intelligence that would ‘learn’ how difficult users found specific questions and then apply a difficulty rating.

## Logical Reasoning

This objective was successfully met in that it was possible to progress to the next round without guessing if a user applied sufficient logic to the problem. The logical reasoning depended on a players cognitive abilities and also the speed at which they could think through the solution. In the early stages there was enough time to simplify the problem by removing wrong answers via bonus balls and also figure out the correct answer. As the levels progressed the time allowed reduced and the amount of available bonus balls also reduced. This was very effective in increasing the difficulty and engagement of the game. Overall all players really liked this feature and it helped that it was familiar to popular games such as wordle that many people had played before. Players especially liked the fact they could colour the year options to help them remember what they thought were correct and incorrect answers (in fact this idea came from a player in the early stages of the game).

Future enhancements could include reversing the setup so that the questions are in the form of 4 years and the answers are in the form of 10 events. This may be more appealing to players in future but this would only become apparent with user testing and evaluation. The program could also have more variety in the bonus features - for example it could have options to tell you which questions you have got the correct answer for - or even remove one of the questions for the level or allow you to replace it with an easier question.

## Reacting Enhancing Features

This was successful but proved to be a much more complicated feature to include. I was very happy with the result and believe the bonus ball moves in a very natural way which was fun to see based on using my A Level physics formulas. The random nature of the bonus ball works very well as it was not predictable where it was going to come from and at what speed/angle. Many players loved the feature and created competitions to see who could get the most bonus balls within the time. This again opened up possibilities for future enhancements of the game. Some players struggled with the mouse as a way of clicking on the bonus ball - however when using the game on a touch screen laptop it proved much easier to capture the ball. Also sometimes the ball disappeared behind other components on the screen making it impossible to click. The biggest problem was the question panel which took up a large part of the screen - an option for the future would be to remove the question panel and just have the individual questions which would reduce this effect.

## Summary

I enjoyed making the game and learned a lot during the process. There were many people who played the game during its development and provided valuable feedback and ideas. I did not manage to successfully create a feedback form and have each of the past players fill it in as it would have been difficult to have everyone take the time to do so. However I feel I gathered a good amount of feedback during the process and this gave me confidence that I had met my original objective.