# Analysis

## Description of the program

My program will be a Tetris-like game, where shapes consisting of four adjacently connected blocks traverse from the top of a 10 x 22 board to the bottom. It can be controlled by moving left, right, rotating clockwise and anticlockwise, making it fall faster and making it fall instantly. The block places after colliding with the bottom of the screen or with another existing block. When an entire row is filled with blocks, the row disappears and all blocks above it moves down, awarding points. The game ends when the area that the pieces appear from is occupied.

My program will have SDD as its main structure and will be object oriented using pygame, which is a python library that allows for a graphical user interface. The program will also integrate with a database, which will store user data and scores. This data will be used to display the top 10 scores from all users by sorting the data in descending order using insertion sort and displayed in a separate menu.

The end users of my program are for people who are already familiar with Tetris and wish to compare how well they play against their friends.

This program will meet the Advanced Higher Computing Science requirements due to the following:

* My program will use a 2d array of objects to store each individual square on the board
* My program will integrate with a database to read in high scores and store it in an array of records
* My program will use insertion sort to sort the scores in descending order for all users
* My program will integrate with a database to read in and insert user data when creating an account and logging in
* The program will also integrate with a database to insert a score after the game ends

## Constraints

These are some of the constraints that will be considered when developing this program:

* Technical constraints:

The hardware used to develop this program:

* + - * Intel family 6 Model 60 Stepping 3 GenuineIntel ~3201 Mhz processor
      * 8098mb ram
      * 9378mb vram

Software used to develop this program:

IDE: Visual Studio Code, Python 3.10 64-bit

Interpreter: Python 3.10 64-bit

Libraries / dependencies: pygame 2.4.0, mysql.connector

OS: Windows 10.0.19042

* Economic considerations:

There will be no costs while developing this program as all software used will be free of charge or open source, and all necessary hardware will be supplied by the school

* Time considerations:

This program will have to be completed and meet all the requirements by the set SQA deadline of the 22nd of March 2024.

## UML case diagram

A diagram of a diagram

Description automatically generated

## End user Requirements:

The end user should be able to:

1. Log in to an existing account and create a new account
2. Select UI on the menus to play, view leaderboards and quit
3. View top 10 scores from all users
4. Be able to play the game
5. Display screen when the game is over

## Functional Requirements:

1. Log in

1. The program must let the user select an option to log in or create a new user.
2. My program must be able to show prompts for users when log in is selected to input their username and password and validate this input by checking for existing users in a database.
3. It must also allow for new users to be created, checking if the user already exists or if the username is between 1 and 32 characters.
4. If the username and password is valid when logging in, the display and the main menu should be initialised

Inputs:

* + - log in/create user option
    - Username and password

Process:

* + - Integrate with a database to check if the user is in the database
    - Check if the password hash matches with the database
    - If the option to create a new user is selected, check if the user already exists, and then create a new account
    - Validate user inputs

Output:

* log in and create user prompts
* Username and password prompts
* Appropriate error message when input is invalid
* Login message

2. Main Menu

1. The program must allow users to select the following options: Play game, open leaderboard and exit.

Inputs:

* UI elements: Play game, leaderboard, exit

Outputs:

* Display window
* Change scene to whatever was selected

3. Leaderboard

a. The program must integrate with the database to retrieve all the scores

b. The program must display the top 10 scores from all users

c. clicking the exit button returns back

Inputs:

* Back to menu button

Processes:

* Sort scores in descending order

Output:

* Top 10 scores

4. Play game

1. The program must be able to read from the settings.json file
2. The program must be able to store and render a 10x22 board
3. The program must be able to store the details of each of the 7 pieces, add them to a queue, create them at the top of the screen, move, rotate and place them.
4. When a line is filled with blocks, the program must recognise this and remove the row. It must also add score to the user when this happens.
5. The game ends when there is a piece at the top of the screen

Inputs:

* + - Settings file
    - Keyboard inputs

Processes:

* Read settings file into an object
* Initialise game variables and piece definitions
* Define actions that trigger when a certain key is pressed, such as: dropping a piece, rotating or moving
* Discard any key inputs that do not have a registered action
* When a piece is moved or rotated, it should check if the squares it will occupy are valid. If it is, then it moves it to the new position. If it isn’t valid, then it will keep the piece in the original position and call other events such as a piece being placed
* Check if a piece is placed and get the next piece from the queue if it has. New pieces should be added when the length of the queue is under 7 pieces.
* It should then check if a row is cleared. If it is, the program should move all rows above it down and add score depending on how many rows have been cleared
* The game should automatically move the piece down 2 times a second
* If the squares where the pieces are created are occupied, the game should end

Outputs:

* Board showing gameplay
* Pieces rendered on the board
* Score

5. Game over

1. The program should display a screen when the game is over and include the user's score, an option to play again and to go back to the menu.
2. The score obtained should be added to the scores table in the database along with the userID and date played

Inputs:

* Mouse pressed for play again and exiting back to the menu

Processes:

* Integrate with a database to add a new score, which includes having to convert the current date into a suitable format

Outputs:

* Suitable game over message
* Score
* Buttons for playing again and leaving

6. general initialisation

1. The program must initialise pygame objects such as clock, display and fonts
2. The program must be able to handle input events such as keyboard, mouse and exit events.
3. The program should allow for UI objects to be created, that have a position, size, colour, text and should have methods that are ran when it has been clicked

Inputs:

* Keyboard and mouse events

Processes:

* Each keyboard event links up to an action that can be detected throughout the program
* Whenever the mouse is pressed, the program should check whether it has clicked a UI element and run it’s onclick method if it has.

Outputs:

* A screen is displayed that the program can communicate with.

7. database

1. This program must contain a database that contains 2 tables: Player, which has the fields playerID, userName and passHash, and Score, which has the fields scoreID, score, datePlayed and the foreign key playerID.
2. The program must be able to create a secure connection with this database
3. The program must also have to be able to run search, update and insert queries

Inputs:

* Fields listed above

Processes:

* Communicate with the database to run selected queries

Outputs:

* Data from the database

## Project plan

|  |  |
| --- | --- |
| Task | Duration (hours) |
| Outline of project | 2 |
| Constraints | 1 |
| UML case diagram | 1 |
| End user requirements | <1 |
| Functional requirements | 4 |
| Project plan | 1 |
| UML class diagram | 2 |
| Pseudocode | 4 |
| Design of data structure and algorithms | 1 |
| Design of database tables and queries | 2 |
| Wireframes / UI design | 3 |
| Database implementation | 2 |
| Software implementation | 80 |
| Research of new skills | 15 |
| Ongoing testing | 7 |
| Test plan | 5 |
| Personas and test cases | 3 |
| Evaluation | 2 |

# Software design

## UML class diagram

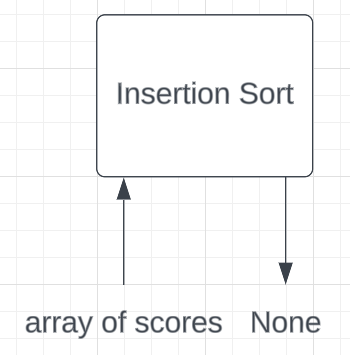
A diagram of a computer

Description automatically generated

A diagram of a computer

Description automatically generated

## Data flow



## Methods

14 classes

### Main:

Main(): initialises all non pygame properties and runs the main loop method

initPg(): initialises properties required for pygame

eventHandle(): processes all pygame inputs such as key and mouse presses

mainLoop(): connects different parts of the code and runs it depending on the current mode: login, game, menu or leaderboard

### Database:

Database(): initialises properties required for the database

connect(): connects to the database, and exits the program if it fails

search(query): searches the database with the given query

mutate(query): adds to or deletes records from the database using the given query

### LogIn:

LogIn(): instantiates the login object with the necessary properties and runs the log in menu

menu(): allows the user to input whether they want to log in or create an account

login(): asks the user for their username and password, hashes the password, compares these values to the database and then either logs the user in or displays an error message.

createUser(): asks the user to enter a username and password, checks if it’s in the database, and if it’s not, it will hash the user’s password and store it in the user table in the database.

### Display:

Display(): instantiates the display object and initialises properties including pygame ui objects

addElement(pos, size, color, text=None): calls the constructor for the UIElement class, and adds it to the ui list

clickEvent(ctx): is called whenever a mouse button is pressed, it loops through all the ui elements to check if they have been pressed, and runs an onClick method if it has been clicked.

render(): loops through all the ui and renders them on to the screen

### UIElement:

UIElement(): instantiates the UIElement object and initialises it’s properties such as size, position, color and text.

bindOnClick(func): sets the onClick method to the given function and sets the ui to be able to be clicked.

onClick(): is defined when bindOnClick is run and is set to the function given in the parameter

### Menu:

Menu(): instantiates the menu object and runs the menuLoop method

menuLoop(): sets up the display to render a main menu, with interactable buttons that allow the user to play the game, view the leaderboard or exit the game.

### Keys:

Keys(): instantiates the keys object, reads the options.json file and creates 2 dictionaries that together describe the key, the action and whether it’s currently pressed

bindOnKey(\*args, \*\*kwargs): defines a wrapper function that converts a key press into an action

### Game:

Game(): instantiates the game object by initialising the necessary properties, and then running the gameLoop method

addAction(\*args): adds an action to the action queue

processActions(): loops through the action queue and runs each action, it also handles placing and generating new pieces

gameLoop(): tetris game code:

* Initialises the board, pieces and ui elements
* Maps the controls to actions such as rotating, placing and moving the piece
* Process actions 16 times a second
* Moves the piece down automatically

### Board:

Board(): instantiates the object and initialises it’s properties, including a 2d array of Square objects

updateBoard(): loops through the 2d array and updates the state of every square

clearRow(rows): checks what rows have been affected by a piece, then loops through those rows to check if it has been filled. If it has, then it removes this row and moves every row above it down by one. It will then add score depending on how many lines have been cleared

### Square:

Square(): instantiates the object and initialises the ui and colour of each square

### Piece:

Piece(): instantiates the piece object and sets the piece position initially to the top of the board

move(direction): moves the piece in the given direction using the following algorithm:

* Calculates the new position by moving the old position by the given direction
* Deletes the squares from the old position from the board
* Checks if the new position is still on the board and if it’s not occupied by another piece
* If the check fails, add the squares from the old position back to the board, and then exits the function with an error message
* If it succeeds add the squares from the new position to the board and updates the board

rotate(direction): rotates the piece either clockwise or anticlockwise depending on the direction, and follows a similar algorithm to the move method

place(): moves the piece all the way to the bottom of the screen, checks what rows the piece occupies and calls the clear row method from the board object with the rows as it’s parameters

### I/O/T/S/Z/J/L Piece:

Constructor(): instantiates the object by inheriting the properties and methods of the Piece class, and initialises the square positions for all the rotations

### Leaderboard:

Leaderboard(): instantiates the object and runs the leaderboardLoop method

leaderboardLoop(): gets all the scores from the database and sorts it using insertion sort. It then renders the ui including the top 10 scores.

insertionSort(scores): runs an insertion sort on an array of scores, sorting in ascending order by the score

### Score:

Score(): instantiates the score class and initialises the name, date and score properties

## Pseudocode:

***Advanced higher criteria***

Data structure

DECLARE scores AS ARRAY OF score INITIALLY []

Insertion sort pseudocode

Define insertionSort method (array of score: scores)

Start a fixed loop i of between 1 and length of scores

initialise int index with value i

initialise a score object called value with value scores[i]

start a conditional loop with the conditions (index > 0 and scores[index-1].score < value.score)

set scores[index] to scores[index-1]

decrement index by 1

end while

set scores[index] to value

End fixed loop

End function

The scores array holds objects from the score class, where each object stores the name, date played and score. An example of the array is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Name | Date | Score |
| 0 | Alice | 3-2-2024 | 2800 |
| 1 | Bob | 8-3-2024 | 5600 |
| 2 | Charlie | 23-12-2024 | 16700 |

Database integration

CLASS Database:

CONSTRUCTOR (main):

DECLARE self.main INITIALLY main

DECLARE self.conn INITIALLY NONE

DECLARE self.cursor INITIALLY NONE

DECLARE self.user INITIALLY –1

END CONSTRUCTOR

METHOD connect():

SET self.conn TO database connection

SET self.cursor TO self.conn.cursor()

CATCH ERROR

DISPLAY error message

END PROGRAM

END EXCEPT

END METHOD

METHOD search(query):

self.cursor.execute(query)

return self.cursor.fetchall()

METHOD mutate(query):

self.cursor.execute(query)

self.conn.commit()

END METHOD

END CLASS

Input validation for login method

DISPLAY “Type ‘back’ to go back”

WHILE TRUE:

DECLARE user INITIALLY KEYBOARD INPUT

IF user == “back”:

RETURN FALSE

END IF

DECLARE passwd INITIALLY KEYBOARD INPUT

HASH passwd

SEARCH DATABASE FOR user AND STORE 2D ARRAY OF playerID, userName, passHash IN data

IF NO DATA:

DISPLAY “user not found”

END IF

ELSE IF data[0][2] == passwd:

SET self.main.db.user TO data[0][0]

SET self.main.mode TO “menu”

DISPLAY “Logged in”

RETURN TRUE

END IF

ELSE:

DISPLAY “Incorrect password”

END ELSE

END WHILE

Input validation for create user method

DISPLAY “type ‘back’ to go back”

EARCH DATABASE FOR ALL userNames AND STORE IN 2D ARRAY OF userName IN usernames

DECLARE taken INITIALLY TRUE

DECLARE name INITIALLY “”

WHILE LENGTH OF name NOT BETWEEN 1 AND 32 OR taken = TRUE:

SET taken TO FALSE

SET name TO KEYBOARD INPUT

IF name == “back”:

RETURN NULL

END IF

FOR EACH user IN usernames:

if user[0] == name:

SET taken TO TRUE

DISPLAY “Username taken”

Break out of for loop

END IF

END FOR

END WHILE

DECALRE passwd INITIALLY KEYBOARD INPUT

HASH passwd

INSERT userName AND passwd INTO player TABLE

***Other***

Game loop

FILL display WITH COLOUR “#FFF8F0”

SET display.ui TO DEFAULT

DECLARE board INITIALLY Board(self)

DECLARE self.score INITIALLY 0

CREATE UI score

DECLARE ticks INITIALLY 0

DECLARE CONSTANT self.pieceList AS [list of every piece]

DECLARE self.currentBag INITIALLY [list of every piece]

RANDOMISE ORDER OF self.currentBag

DECLARE self.currentPiece INITIALLY self.currentBag[0]

DECLARE ACTION hDrop AS self.currentPiece.place

DECLARE ACTION rotClock AS currentPiece.rotate(1)

DECLARE ACTION rotAnti AS currentPiece.rotate(-1)

WHILE GAME IS RUNNING:

DRAW BOARD

IF down PRESSED:

MOVE CURRENT PIECE DOWN

END IF

IF left PRESSED:

MOVE CURRENT PIECE LEFT

END IF

IF right PRESSED:

MOVE CURRENT PIECE RIGHT

END IF

DISPLAY SCORE

UPDATE BOARD

UPDATE DISPLAY

GET INPUTS

WAIT FOR 1/ self.main.tickrate seconds

SET ticks TO ticks + 1

RUN EVERY 1/16 seconds:

self.processActions()

END IF

RUN every 1/2 seconds:

MOVE CURRENT PIECE DOWN

END IF

END WHILE

TRIGGER game over

return 0

## User Interface

Log in

|  |  |  |
| --- | --- | --- |
| Display | Action | Process |
| 1. Login  2. Create User  [input] | Keyboard Input | Validate input and call login() or createUser() method |
| If 1 is pressed:  Type “back” to go back  Username: [input]  Password: [input] | Keyboard Input | Check if username and password matches with the database |
| If username is invalid:  User not found  Username: [input]  Password: [input] | Keyboard Input | Same processes and validation as above |
| If password is invalid:  Incorrect password  Username: [input]  Password: [input] | Keyboard Input | Same processes and validation as above |
| If username and password is valid:  Logged in |  | Load main menu |
| If “back” is typed:  1. Login  2. Create User  [input] | Keyboard Input | Returns to the login menu |
| If 2 is pressed:  Type "back" to go back  Enter a username between 1-32 characters [input] | Keyboard Input | Validates username |
| If username is taken:  Username taken  Enter a username between 1-32 characters [input] | Keyboard Input | Same validation as above |
| If username is invalid:  Enter a username between 1-32 characters [input] | Keyboard Input | Same validation as above |
| If username is valid:  Enter a password  [input] | Keyboard Input | Hashes the given password |
| When the user enters a password:  User has been created  1. Login  2. Create User  [input] | Keyboard Input | Returns to the login menu |

Main Menu

A screenshot of a computer

Description automatically generated

Game

A screenshot of a graph

Description automatically generated

Game Over

A screenshot of a game

Description automatically generated

Leaderbaord

A screenshot of a computer

Description automatically generated

# Database design

## Data dictionary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Entity: Player | | | | | |
| Field name | Key | Type | Size | Required | Validation |
| playerID | PK | int |  | yes | Auto increment |
| userName |  | varchar | 32 | yes |  |
| passHash |  | varchar | 64 | yes |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Entity: Score | | | | | |
| Field name | Key | Type | Size | Required | Validation |
| scoreID | PK | int |  | yes | Auto increment |
| score |  | int |  | yes |  |
| datePlayed |  | date |  | yes |  |
| playerID | FK | int |  | yes | Existing playerID from Player table |

## Entity relationship diagram

A diagram of a diagram

Description automatically generated

## Queries

Log In:

Get specific user data

|  |  |
| --- | --- |
| Field(s)/Calculation(s) | PlayerID, userName, passHash |
| Table(s)/Query(ies) | player |
| Search Criteria | UserName = [user] |

Get all users

|  |  |
| --- | --- |
| Field(s)/Calculation(s) | userName |
| Table(s)/Query(ies) | player |
| Search Criteria |  |

Create user

|  |  |
| --- | --- |
| Insert | UserName, passHash |
| Table | player |
| Values | [Name], [passwd] |

Game over:

Add score

|  |  |
| --- | --- |
| Insert | Score, datePlayed, playerID |
| Table | score |
| Values | [Self.score], [currentDate], [self.main.db.user] |

Leaderboard:

Get all scores

|  |  |
| --- | --- |
| Field(s)/Calculation(s) | username, datePlayed, score |
| FROM Table(s)/Query(ies) | Score, player |
| Search Criteria |  |

# Implementation

## Insertion sort

(Found in leaderboard.py)

A computer code with text

Description automatically generated with medium confidenceA computer code with text

Description automatically generated with medium confidence





## Array of records

(Found in leaderboardLoop(), leaderboard.py)

A black text on a white background

Description automatically generated

## Input validation

(found in logIn.py)

A screenshot of a computer code

Description automatically generated

## Database

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

## Database connection

(found in db.py)

A screen shot of a computer code

Description automatically generated

## Database queries



A close-up of a number

Description automatically generated



A close-up of a white background

Description automatically generated



A number on a white background

Description automatically generated



A close up of a date

Description automatically generated





## UI

(found in logIn.py)

A screen shot of a computer screen

Description automatically generated

A screenshot of a computer error

Description automatically generated

(found in menu.py)

A screenshot of a computer

Description automatically generated

(found in game.py)

A screenshot of a game

Description automatically generated

(found in gameOver.py)

A screenshot of a game

Description automatically generated

(found in leaderboard.py)

A screenshot of a computer

Description automatically generated

## Skills learned

|  |  |  |
| --- | --- | --- |
| Skill | Research | Application |
| Pygame | The pygame.org’s documentation for pygame (https://www.pygame.org/docs/index.html) | Used for the front end of the project. Without this, the program wouldn’t be able to display anything to the user or get any keyboard inputs |
| Hashing a string using the hashlib library | Python’s library reference  <https://docs.python.org/3/library/hashlib.html>  Specifically: | Used to hash the user’s password to store it securely. |

## Encountered errors

|  |
| --- |
| (in ui.py)  A computer code with black text  Description automatically generated  The variable pos was used in the declaration of self.surface instead of size.  A computer code with black text  Description automatically generated |
| (in pieces.py)  A white background with black text and numbers  Description automatically generated  A 2d array was used instead of a 3d array to store the piece data  A white background with black text  Description automatically generated |
| (in clearRow(), game.py)  The loop looped through the columns of the board instead of the rows  A black text on a white background  Description automatically generated |
| (in clearRow(), game.py)  A white background with black and red text  Description automatically generated  When the row moves down a new row doesn’t get created at the top of the screen, leads to an interesting visual error  **A colorful rectangles with different colors  Description automatically generated**  A white background with black text  Description automatically generated |
| (in login(), logIn.py)  A white background with colorful text  Description automatically generated  The program couldn’t log in with the correct password because the password wasn’t hashed  A computer screen shot of text  Description automatically generated |
| (in processActions(), game.py)  A screen shot of a computer code  Description automatically generated  The variable i was redefined later on in the loop, causing there to be approximately a 1 in 7 chance for a line to not clear.  A yellow purple and white squares  Description automatically generated  A screen shot of a computer code  Description automatically generated |
| (in processActions(), game.py)    The program did not exit the loop after placing a piece, making it possible for a piece to float when moved at the same time as it being placed  A red and yellow squares  Description automatically generated  A black and white text  Description automatically generated |

# Testing

## Testing plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Objective | Description | Expected result |
| Functional requirement 1 | | | |
| 1.a.i | Test that the program displays the log in screen when that option is selected | Run the login menu method and type “1” into the prompt | The program runs the login method |
| 1.a.ii | Test that the program displays the create user screen when that option is selected | Run the login menu method and type”2” into the prompt | The program runs the create user method |
| 1.a.iii | Test the program for when a different option than those that were given was submitted | Run the login menu method and type an invalid input | The program tells the user the menu prompts again |
| 1.b.i | Test that the program can search the database for a specific user from the player table, and returns the playerID, userName and passHash | Run a modified version of the login method that displays the output of the database query | The program should display the playerID, userName and passHash of a given username that is identical to a record in the player table |
| 1.b.ii | Test that the program can log in given a valid username and password | Run the login method and input a valid username and password | The program displays a message telling the user that they are logged in |
| 1.b.iii | Test that the program returns a valid error message when the user doesn’t exist | Run the login method and input the username of a user that doesn’t currently exist in the database | The program tells the user that the specified user doesn’t exist and asks the user to type in a username and password again |
| 1.b.iv | Test that the program returns a valid error message when an incorrect password is submitted | Run the login method and input a valid username but an incorrect password | The program tells the user that the password is incorrect and asks the user to type in a username and password again |
| 1.b.v | Test that the program returns to the menu when “back” is typed as it’s username | Run the login method and type “back” into the prompt | The program returns to the login/create user menu |
| 1.c.i | Test that the program can search the database for all users from the player table | Run a modified version of the create user method that displays the output of the select database query | The program should display all usernames from the player table and should match the database |
| 1.c.ii | Test that the program allows a username that has between 1 and 32 characters and is not already taken | Run the create user method and enter a username that is valid | The program asks the user for a password |
| 1.c.iii | Test that the program invalidates a username that is less than 1 character in length | Run the create user method and leave the field blank | The program asks the user to input a username between 1 and 32 characters again |
| 1.c.iv | Test that the program  invalidates a username that is more than 32 characters | Run the create user method and enter a username greater than 32 characters | The program asks the user to input a username between 1 and 32 characters again |
| 1.c.v | Test that the program invalidates a username that is already taken | Run the create user method and enter a username that already exists in the database | The program tells the user that the username is already taken, and asks for a new username |
| 1.c.vi | Test that the program returns to the menu when “back” is typed as it’s username | Run the create user method and type “back” into the prompt | The program returns to the login/create user menu |
| 1.c.vii | Test that the program hashes the password correctly | After typing in a valid username, enter a password | The program should calculate a hashed version of the given password that cannot be used to find the original password |
| 1.c.viii | Test that the program can add a new user to the user table | Run the create user method and enter a valid username and password | A new instance of the user should be created in the user table in the database |
| 1.d.i | Test that the main menu class is initialised and run when the user logs in | Run the log in method and input a valid username and password | A new window should be created displaying the main menu. |
| Functional requirement 2 | | | |
| 2.a.i | Test that clicking the play button results in the game being played | Click the play button on the main menu | The game starts |
| 2.a.ii | Test that clicking the leaderboard button results in the leaderboard being displayed | Click the leaderboard button on the main menu | The leaderboard screen is displayed |
| 2.a.iii | Test that clicking the exit button results in the program quitting | Click the exit button on the main menu | The program quits |
| Functional requirement 3 | | | |
| 3.a.i | Test that the program can search the database and retrieve all scores | Modify the leaderboard loop method such that the results from the select statement is returned | The program displays the username, dateplayed and score for all records in the score and player table and should match with the database |
| 3.b.i | Test that the program can run an insertion sort on an array of records | Create a driver that allows for custom user scores to be used and display the array before and after the sort | The program displays the contents of the array and then a sorted version of the same data |
| 3.b.ii | Test that the program displays a maximum of 10 scores on the screen | Populate the database with more than 10 scores and open the leaderboard menu | The program should only display 10 scores |
| 3.b.iii | Test that the program can display less than 10 scores | Populate the database with less than 10 scores and open the leaderboard menu | The program should display all the scores from the score table and fill the rest of the table with blank space |
| 3.c.i | Test that pressing the exit button returns the user back to the menu | Click the exit button on the leaderboard menu | The program should return to the main menu. |
| Functional requirement 4 | | | |
| 4.a.i | Test that the data read from the settings.json file is stored correctly | Modify the keys class to display the data structure storing the controls | The program should display what keys have been assigned to which action |
| 4.a.ii | Test that invalid data read from the settings file results in the program terminating | Input incorrect data into the settings file | The program should stop after logging in with a message saying that the controls are invalid |
| 4.b.i | Test that the size of the board is 10x22 | Modify the board class to display the size of the 2d array | The program displays that the board is 10x22 |
| 4.b.ii | Test that each square can store it’s state | Modify the game class to change one of the squares to a different colour and display the colour of each square | The program should display that all the squares but 1 are empty |
| 4.b.iii | Test that the board can be displayed | Run the game | The program should display every square in the 2d array in a 10x22 grid. |
| 4.c.i | Test that all the pieces are the correct shape | Run the game and place the first 7 pieces | The program should place 7 pieces of different colours that corresponds to its own shape |
| 4.c.ii | Test that all the pieces follow a system where each piece is added to a queue and is chosen only once every 7 pieces | Run the game and place the first 7 pieces | The program should place 7 different pieces |
| 4.c.iii | Test that new pieces are added to the queue when the length of the queue is under 7 pieces | Modify the game loop method to display the length of the queue whenever a piece is placed | The program should display that the length of the queue is initially 7, and then 13 after a piece is placed |
| 4.c.iv | Test that the piece can move when a corresponding key is pressed | Run the game and press left, then right, then down, then space | The program should move the piece left, then right, then down, then to the bottom of the screen |
| 4.c.v | Test that the piece can rotate clockwise and anticlockwise | Run the game and press x, then z | The program should rotate the piece clockwise, then anticlockwise |
| 4.c.vi | Test that when the piece cannot move or rotate a certain way, it doesn’t move or rotate | Run the game and move the piece to the edge of the screen, then press rotate | The program should keep the piece on the screen and not let it rotate |
| 4.c.vii | Test that when the piece hits the bottom or another piece, it places | Run the game and place a piece, then place another piece on top of it | The program should create a new piece when the first piece touches the ground, then a third when the second one touches the first piece |
| 4.c.viii | Test that the piece moves down once every 0.5 seconds | Run the game and pull out a stopwatch | The program should move the piece down every 0.5 seconds |
| 4.d.i | Test that the program can detect whenever a line is cleared | Run the game and fill an entire row | The program should remove that line and move everything above it down by one |
| 4.d.ii | Test that the program rewards the correct number of points based on how many lines were cleared at once | Run the game and clear 1, 2, 3 and 4 rows | The program should add 40, 100, 300, 1200 to the score. |
| 4.e.i | Test that the program ends when it tries to create a piece in a place that is already occupied | Run the game and place pieces until it reaches the top | The game over screen should pop up |
| Functional requirement 5 | | | |
| 5.a.i | Test that the program displays a screen with a message saying the game is over, the users score and option to play again and an option to go back to the menu | Run the game over menu | The game over screen should display a game over message, the users score, an option to play again and an option to go back to the menu |
| 5.a.ii | Test that the play again button starts a new game | Run the game over menu and click the play again button | The program should start a new game |
| 5.a.iii | Test that the exit button returns the user back to the main menu | Run the game over menu and click the menu button | The program should open the main menu |
| 5.b.i | Test that the correct date is generated correctly | Modify the game over menu so that the date generated is displayed | The program should display the correct date |
| 5.b.ii | Test that the program can add a new score to the score table | Run the game over menu | A new score should be created in the score table with valid data |
| Functional requirement 6 | | | |
| 6.a.i | Test that the program initialises the clock, display and fonts objects from the pygame module | Modify the main file to display all the variables related to pygame | The program should display all the variables, meaning that they have been initialised |
| 6.b.i | Test that the program can detect when a key that has an action is pressed | Modify the event handle method to output whenever a key is pressed, then press the right key and the a key | The program should display that the right key is pressed and then that a key that doesn’t have an action was pressed |
| 6.b.ii | Test that the program can detect that a mouse button was pressed | Modify the event handle method to output whenever a mouse button is pressed, then click the screen | The program should display that the mouse has been pressed |
| 6.c.i | Test that the program can display ui elements with custom text, size and position | Run the main menu | The program should display buttons |
| 6.c.ii | Test that the ui elements run their onclick function when the left mouse button is pressed down and then up again | Run the main menu and hold click over the play button and release it on the button | The program should start the game |
| 6.c.iii | Test that the ui elements do not run their onclick function when the button is pressed down, and is released outside the button | Run the main menu and hold click over the play button and it release it off to the side | Nothing |
| 6.c.iv | Test that the ui elements do not run their onclick function when the left click is pressed outside the button and then released inside the button | Run the main menu and hold click outside the play button and then release it over the play button | Nothing |
| Functional requirement 7 | | | |
| 7.a.i | Test that the database contains the correct tables with the correct fields | Open the SQL database | There should be only 2 tables: The player table should have the fields playerID, username and passHash, and the score table should have the fields scoreID, score, datePlayed, and PlayerID |
| 7.b.i | Test that the program can connect to the database | Run the program | The program should ask the user to choose between logging in and signing up |
| 7.b.ii | Test that the program exits when it can’t connect to the database | Run the program without the database running | The program displays a valid error message and quits |

## Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Objective | Description | Expected Result | Actual Result |
| 1.a.i | Test that the program displays the log in screen when that option is selected | Run the login menu method and type “1” into the prompt | The program runs the login method | The program runs the login method |
| 1.a.ii | Test that the program displays the create user screen when that option is selected | Run the login menu method and type”2” into the prompt | The program runs the create user method | The program runs the create user method |
| 1.a.iii | Test the program for when a different option than those that were given was submitted | Run the login menu method and type an invalid input | The program tells the user the menu prompts again |  |
| 1.b.i | Test that the program can search the database for a specific user from the player table, and returns the playerID, userName and passHash | Run a modified version of the login method that displays the output of the database query, and then type in alice and password | The program should display the playerID, userName and passHash of a given username that is identical to a record in the player table |  |
| 1.b.ii | Test that the program can log in given a valid username and password | Run the login method and input alice and password | The program displays a message telling the user that they are logged in |  |
| 1.b.iii | Test that the program returns a valid error message when the user doesn’t exist | Run the login method and input the username of a user that doesn’t currently exist in the database e.g. Charlie | The program tells the user that the specified user doesn’t exist and asks the user to type in a username and password again |  |
| 1.b.iv | Test that the program returns a valid error message when an incorrect password is submitted | Run the login method and input a valid username but an incorrect password e.g. alice and abc123 | The program tells the user that the password is incorrect and asks the user to type in a username and password again | A computer screen shot of a password  Description automatically generated |
| 1.b.v | Test that the program returns to the menu when “back” is typed as it’s username | Run the login method and type “back” into the prompt | The program returns to the login/create user menu | A white background with black text  Description automatically generated |
| 1.c.i | Test that the program can search the database for all users from the player table | Run a modified version of the create user method that displays the output of the select database query | The program should display all usernames from the player table and should match the database |  |
| 1.c.ii | Test that the program allows a username that has between 1 and 32 characters and is not already taken | Run the create user method and enter a username that is valid e.g. alice | The program asks the user for a password |  |
| 1.c.iii | Test that the program invalidates a username that is less than 1 character in length | Run the create user method and leave the field blank | The program asks the user to input a username between 1 and 32 characters again | A close-up of blue text  Description automatically generated |
| 1.c.iv | Test that the program  invalidates a username that is more than 32 characters | Run the create user method and enter a username greater than 32 characters | The program asks the user to input a username between 1 and 32 characters again | A close-up of numbers  Description automatically generated |
| 1.c.v | Test that the program invalidates a username that is already taken | Run the create user method and enter a username that already exists in the database | The program tells the user that the username is already taken, and asks for a new username | A white background with blue text  Description automatically generated |
| 1.c.vi | Test that the program returns to the menu when “back” is typed as it’s username | Run the create user method and type “back” into the prompt | The program returns to the login/create user menu |  |
| 1.c.vii | Test that the program hashes the password correctly | After typing in a valid username, enter a password e.g. password | The program should calculate a hashed version of the given password that cannot be used to find the original password and display it |  |
| 1.c.viii | Test that the program can add a new user to the user table | Run the create user method and enter a valid username and password e.g. alice and password | A new instance of the user should be created in the user table in the database | A close-up of blue text  Description automatically generated  A number on a white background  Description automatically generated |
| 1.d.i | Test that the main menu class is initialised and run when the user logs in | Run the log in method and input a valid username and password | A new window should be created displaying the main menu. | A screen shot of a computer screen  Description automatically generated  A screenshot of a computer  Description automatically generated |
| Functional requirement 2 | | | | |
| 2.a.i | Test that clicking the play button results in the game being played | Click the play button on the main menu | The game starts | A screenshot of a game  Description automatically generated |
| 2.a.ii | Test that clicking the leaderboard button results in the leaderboard being displayed | Click the leaderboard button on the main menu | The leaderboard screen is displayed | A screenshot of a computer  Description automatically generated |
| 2.a.iii | Test that clicking the quit button results in the program exiting | Click the quit button on the main menu | The program quits | A screenshot of a computer screen  Description automatically generated  A computer screen shot of a computer code  Description automatically generated |
| Functional requirement 3 | | | | |
| 3.a.i | Test that the program can search the database and retrieve all scores | Modify the leaderboard loop method such that the results from the select statement is displayed | The program displays the username, dateplayed and score for all records in the score and player table and should match with the database | A screenshot of a computer  Description automatically generated |
| 3.b.i | Test that the program can run an insertion sort on an array of records | Create a driver that allows for custom user scores to be used and display the array before and after the sort | The program displays the contents of the array and then a sorted version of the same data | A computer code with text  Description automatically generated with medium confidence |
| 3.b.ii | Test that the program displays a maximum of 10 scores on the screen | Populate the database with more than 10 scores and open the leaderboard menu | The program should only display 10 scores | A screenshot of a computer  Description automatically generated |
| 3.b.iii | Test that the program can display less than 10 scores | Populate the database with less than 10 scores and open the leaderboard menu | The program should display all the scores from the score table and fill the rest of the table with blank space | A screenshot of a computer  Description automatically generated |
| 3.c.i | Test that pressing the exit button returns the user back to the menu | Click the exit button on the leaderboard menu | The program should return to the main menu. | A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated |
| Functional requirement 4 | | | | |
| 4.a.i | Test that the data read from the options.json file is stored correctly | Modify the keys class to display the data structure storing the controls | The program should display what keys have been assigned to which action | hDrop, rotClock, and rotAnti use their corresponding Unicode character. Left, right and sDrop use the pygame constant for the left, right and down key  A computer code with black text  Description automatically generated |
| 4.a.ii | Test that invalid data read from the settings file results in the program terminating | Input incorrect data into the settings file | The program should stop after logging in with a message saying that the controls are invalid | A computer code with black text  Description automatically generated  A screen shot of a computer screen  Description automatically generated |
| 4.b.i | Test that the size of the board is 10x22 | Modify the board class to display the size of the 2d array | The program displays that the board is 10x22 |  |
| 4.b.ii | Test that each square can store it’s state | Modify the game class to change one of the first square to a different colour and display the colour of each square | The program should display that all the squares but the first are empty | A close up of a computer screen  Description automatically generated |
| 4.b.iii | Test that the board can be displayed | Run the game | The program should display every square in the 2d array in a 10x22 grid. | A white rectangular object with a yellow border  Description automatically generated |
| 4.c.i | Test that all the pieces are the correct shape | Run the game and place the first 7 pieces | The program should place 7 pieces of different colours that corresponds to its own shape | A screenshot of a video game  Description automatically generated |
| 4.c.ii | Test that all the pieces follow a system where each piece is added to a queue and is chosen only once every 7 pieces | Run the game and place the first 7 pieces | The program should place 7 different pieces | A screenshot of a video game  Description automatically generated |
| 4.c.iii | Test that new pieces are added to the queue when the length of the queue is under 7 pieces | Modify the game loop method to display the length of the queue whenever a piece is placed | The program should display that the length of the queue is initially 7, and then 13 after a piece is placed |  |
| 4.c.iv | Test that the piece can move when the corresponding key is pressed | Run the game and press left, then down,  then right, then space | The program should move the piece left,  then down, then right, then to the bottom of the screen | A white rectangular object with a yellow border  Description automatically generatedA white rectangular object with a yellow border  Description automatically generated  A white rectangular object with a white background  Description automatically generatedA white screen with a yellow border  Description automatically generated  A white rectangular object with a yellow border  Description automatically generated |
| 4.c.v | Test that the piece can rotate clockwise and anticlockwise | Run the game and press x, then z | The program should rotate the piece clockwise, then anticlockwise | A white rectangular object with a yellow border  Description automatically generated A white screen with a pink border  Description automatically generated with medium confidenceA white rectangular object with a yellow border  Description automatically generated |
| 4.c.vi | Test that when the piece cannot move or rotate a certain way, it doesn’t move or rotate | Run the game and move the piece to the edge of the screen, then press rotate | The program should keep the piece on the screen and not let it rotate | A white rectangular object with a yellow border  Description automatically generatedA white rectangular object with a yellow border  Description automatically generatedA white rectangular object with a yellow border  Description automatically generatedA white rectangular object with a yellow border  Description automatically generated |
| 4.c.vii | Test that when the piece hits the bottom or another piece, it places | Run the game and place a piece, then place another piece on top of it | The program should create a new piece when the first piece touches the ground, then a third when the second one touches the first piece | A white rectangular object with a yellow border  Description automatically generatedA white rectangular object with a yellow border  Description automatically generatedA white rectangular object with a yellow border  Description automatically generated |
| 4.c.viii | Test that the piece moves down once every 0.5 seconds | Run the game and do nothing | The program should move the piece down by itself | A white rectangular object with a white border  Description automatically generatedA white rectangular object with a white border  Description automatically generated |
| 4.d.i | Test that the program can detect whenever a line is cleared | Run the game and fill an entire row | The program should remove that line and move everything above it down by one | A white rectangular object with a white background  Description automatically generatedA white rectangular object with a red border  Description automatically generated |
| 4.d.ii | Test that the program rewards the correct number of points based on how many lines were cleared at once | Run the game and clear 1, 2, 3 and 4 rows | The program should add 40, 100, 300, 1200 to the score. | A white rectangular object with a yellow border  Description automatically generatedA white rectangular frame with orange and white border  Description automatically generated  A screenshot of a video game  Description automatically generatedA white rectangular object with a yellow border  Description automatically generated  A screen shot of a phone  Description automatically generatedA white rectangular object with a yellow border  Description automatically generated  A screen shot of a screen  Description automatically generatedA white rectangular object with a white border  Description automatically generated |
| 4.e.i | Test that the program ends when it tries to create a piece in a place that is already occupied | Run the game and place pieces until it reaches the top | The game over screen should pop up | A colorful blocks on a white background  Description automatically generatedA screenshot of a game  Description automatically generated |
| Functional requirement 5 | | | | |
| 5.a.i | Test that the program displays a screen with a message saying the game is over, the users score and option to play again and an option to go back to the menu | Run the game over menu | The game over screen should display a game over message, the users score, an option to play again and an option to go back to the menu | A screenshot of a game  Description automatically generated |
| 5.a.ii | Test that the play again button starts a new game | Run the game over menu and click the play again button | The program should start a new game | A screenshot of a game  Description automatically generated  A white rectangular object with a white rectangle  Description automatically generated |
| 5.a.iii | Test that the exit button returns the user back to the main menu | Run the game over menu and click the menu button | The program should open the main menu | A screenshot of a game  Description automatically generated  A screenshot of a game  Description automatically generated |
| 5.b.i | Test that the correct date is generated correctly | Modify the game over menu so that the date generated is displayed | The program should display the correct date |  |
| 5.b.ii | Test that the program can add a new score to the score table | Run the game over menu | A new score should be created in the score table with valid data | A screenshot of a game  Description automatically generatedA close up of a date  Description automatically generated |
| Functional requirement 6 | | | | |
| 6.a.i | Test that the program initialises the clock, display and fonts objects from the pygame module | Modify the main file to display all the variables related to pygame | The program should display all the variables, meaning that they have been initialised |  |
| 6.b.i | Test that the program can detect when a key that has an action is pressed | Modify the event handle method to output whenever a key is pressed, then press the space key and the a key | The program should display the unicode for space and then that a key that doesn’t have an action was pressed | A computer code with numbers and symbols  Description automatically generated with medium confidence |
| 6.b.ii | Test that the program can detect that a mouse button was pressed | Modify the event handle method to output whenever a mouse button is pressed, then click the screen | The program should display that the mouse has been pressed | A close up of a text  Description automatically generated |
| 6.c.i | Test that the program can display ui elements with custom text, size and position | Run the main menu | The program should display buttons | A screenshot of a game  Description automatically generated |
| 6.c.ii | Test that the ui elements run their onclick function when the left mouse button is pressed down and then up again | Run the main menu and hold click over the play button and release it on the button | The program should start the game | A screenshot of a game  Description automatically generatedA screen shot of a phone  Description automatically generated |
| 6.c.iii | Test that the ui elements do not run their onclick function when the button is pressed down, and is released outside the button | Run the main menu and hold click over the play button and it release it off to the side | Nothing | A screenshot of a game  Description automatically generatedA screenshot of a game  Description automatically generated |
| 6.c.iv | Test that the ui elements do not run their onclick function when the left click is pressed outside the button and then released inside the button | Run the main menu and hold click outside the play button and then release it over the play button | Nothing | A screenshot of a game  Description automatically generatedA screenshot of a game  Description automatically generated |
| Functional requirement 7 | | | | |
| 7.a.i | Test that the database contains the correct tables with the correct fields | Open the SQL database | There should be only 2 tables: The player table should have the fields playerID, username and passHash, and the score table should have the fields scoreID, score, datePlayed, and PlayerID | A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated |
| 7.b.i | Test that the program can connect to the database | Modify the connect method to display when the database has successfully connected and then run the program | The program should display that it has connected to the database | A close-up of a white background  Description automatically generated |
| 7.b.ii | Test that the program exits when it can’t connect to the database | Run the program without the database running | The program displays a valid error message and stops running |  |

## Persona Testing

|  |  |  |
| --- | --- | --- |
| Persona | Test plan | Feedback |
| Tom is an s6 student that is familiar with the game of tetris, however isn’t great with text based user interface | The user should create a user, log in and play a game. After finishing a game, they should load the leaderboard and then exit out the program | The user interface was intuitive, and he navigated each menu without difficulty. The control layout was awkward at first but quickly became natural. The movement of the pieces was “too sensitive” and the game would’ve benefitted with gridlines to tell where the pieces would drop. |
| Jack is an s6 student that is very familiar with tetris and has a deep understanding of game development | The user should create a user, log in and view the leaderboard. They should then play a game and then exit out the program | The text based user interface did not pose any problems. The user did mention that there were many features missing that are present in most tetris games, such as viewing upcoming pieces and storing pieces. |
| Will is an s6 student that is somewhat familiar with tetris and is computer literate | The user should create a user with an invalid username, then create a valid user, log in and view the leaderboard. Then they should play a game and exit the program | The user liked the robustness and the ease of use of the text based ui, and was impressed by the simplicity of the program. Some complaints the user had was that the pieces were “slippery” and that the game didn’t increase the speed when the score increased. |

# Evaluation

## Fitness for purpose

The purpose of this project was to create a Tetris inspired game that had to have a log in system that could allow a user to create a user and integrate with a database to store it, and log in, while also validating the users inputs. It had to contain a main menu that allowed the user to view a leaderboard, and play a game, that had a game over screen and would integrate with a database to store the scores.

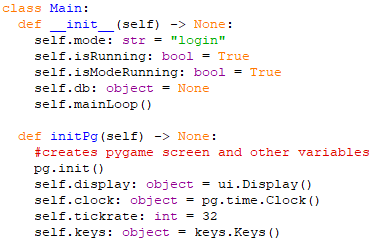
The tests that have been run during final testing showed that the project has met all the end user and functional requirements that were originally set during the analysis stage. The personas managed to complete the test cases given to them without much difficulty

|  |  |
| --- | --- |
| Original requirement | Completed |
| The program must allow the user to create an account and log in | *✓* |
| The program must display a menu that allows the user to play the game, view the leaderboard and exit the game | *✓* |
| The program must display a leaderboard that shows the top 10 scores | *✓* |
| The program must be able to play tetris | *✓* |
| The program must display a game over screen when the game is over | *✓* |
| The program must initialise the pygame library and use it to display a user interface | *✓* |
| The program must integrate with a database to create users and scores, and be able to retrieve this information | *✓* |

## Maintainability

My program is maintainable because I have split my program into different files, classes and subroutines, which allows for efficient corrective maintenance. So if an error is found by the user, it can be quickly located in the program without struggling to find which part of the code is responsible for the error. I have also used internal commentary that briefly describes complex parts of the code which reduces the time spent figuring out what each line of code does. An example of these is shown below:

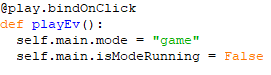
(found in main.py)



Because my program uses subroutines, it reduces the time spent on perfective maintenance. This is because adding new features and menus are created from reusing code used earlier in the project instead of rewriting code again:

(found in menuLoop(), menu.py)

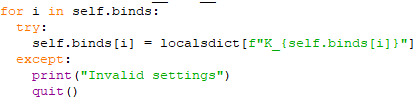




## Robustness

My program is robust as all user inputs is validated by the program, including the options.json file. Inputting invalid data into the file will stop the program from running and prevent code that the user might input into the file from executing:

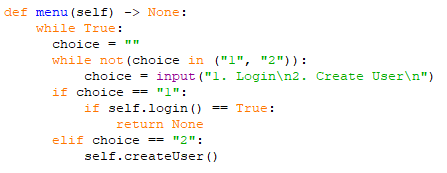
(found in keys.py)



When the user writes something that doesn’t correspond to a key into the file, the program recognises this and closes the program.

When the program is run, all inputs in the text interface are validated, making it impossible for the user to accidently break the program by typing in data that the program cannot handle.

(found in logIn.py)



In this example, the program asks the user to either log in or create a user. If neither option is typed, the program handles this by displaying the options again and allowing the user to input an option again.