

Analysis

The problem

My client is a full-time primary school teacher who currently teaches a full class of year 4 primary school students. She teaches then a variety of subjects, but her main subject area of expertise is mathematics. She also tutors a small group of students outside of school on a weekly basis to help improve their maths skills.

The teacher is also an innovative educator who believes that introducing computer games into education process will revolutionise education, due to the increase in technology used in our generation. As a result, she is very keen to start using games with her students. She is currently concerned about the mental maths skills of the students she teaches, and would like to use such games to help them. The problem she has is which type of game she will actually start from that would help her to improve the mental maths skills of the students she teaches.

She would like me to produce an interactive desktop game for her students. She wants the game to have an educational and mathematical aspect with a scoring system to help motivate her students to develop their mental maths and critical thinking skills by playing this game.

I have suggested for the teacher to use game based on Tetris to help to solve her problem, a well-known tile matching video game with many versions and released on many platforms. The idea is suitable since it is appropriate for the age group of her students, and it complements her specialisation in maths well. The teacher has very much agreed with this idea.

As an extra feature, she would also like the game to 'remember' the high scores gained by the user to be able to view them anytime the game is played.

To suit the teacher's needs, it will have a unique feature that involves an educational aspect in order to distinguish it from other Tetris variants.

There will be potential users of around the same age as my target audience involved in the project, in order to get feedback from them about the game.

Tetris Guideline: Design for Interface

The guideline document also consists of the standard appearance for the game interface. The numbered labels shown in the image represent the following:

1- **The playfield**, where the game play actually occurs.

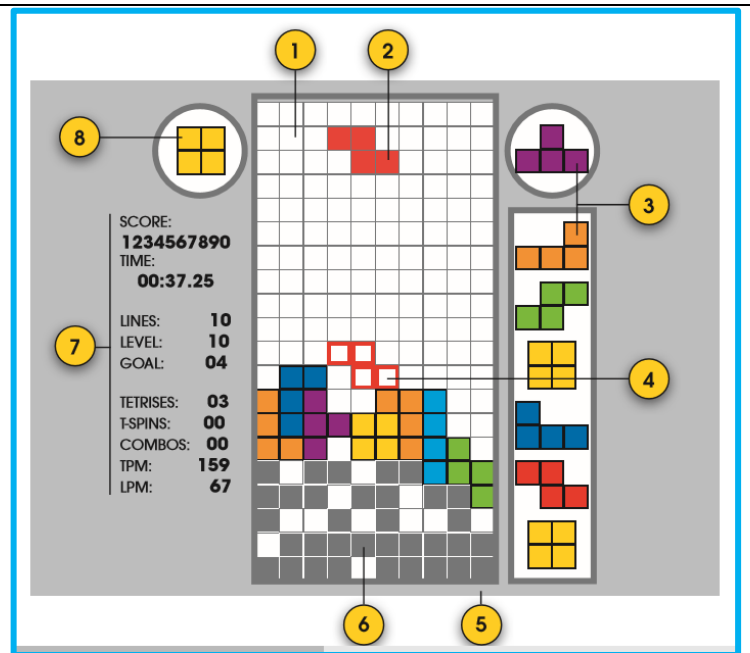
2- **The active piece**, the tetromino in play which can be controlled by the player.

3- **The Next Queue**, showing the next piece that will be put into play, as well as the pieces after. The Guideline specifies for at least one and no more than six pieces in the next queue. It also says for the component to be placed at the top-right of the playfield.

4- **The Ghost Piece**, the translucent image of the active piece, indicating its position if it were to be dropped directly below its current position. It should be allowed to be toggled on or off.

5- The **background graphic** for the interface (***not very essential***)

6- **Starting blocks**, present in some variations of Tetris, where the player can vary the number of rows of blocks at the bottom of the playfield that the game should start with (***not an essential feature***)



7- **Game Information**, showing information that is relevant to the current game session being played. The set of fields shown varies between Tetris versions. The most common fields are:

- Current Score
- High Score
- Number of lines cleared
- Current Level

Other possible fields are:

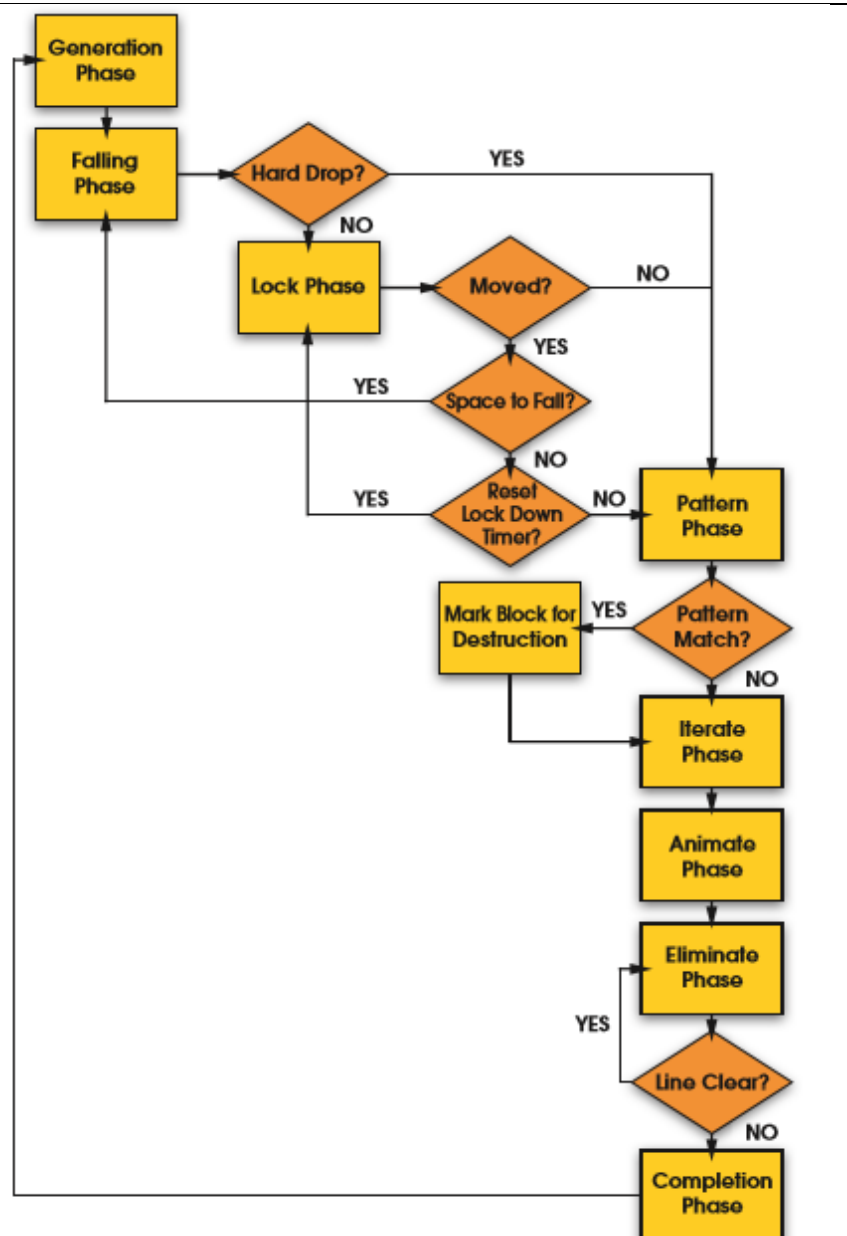
- Time elapsed
- Game mode
- Player name

8- **Hold Queue**, which allows the player to store an active piece as long as necessary. If piece is already inside, it will swap places with the active piece. The guideline states for it to be displayed on the top-left of the playfield.

Tetris Guideline: Tetris Engine Flowchart

The Tetris Engine is the code that controls the logic of a Tetris game. The guideline includes a flowchart as shown below which describes the game logic and the phases the 8 active tetromino goes through in the Tetris Engine as the game is being played.

1. **Generation Phase**: This phase involves generating a tetromino and placing it onto the playfield.
2. **Falling Phase**: The phase where the player is allowed to move, rotate, soft drop and hard drop the tetromino. Once it lands on a surface, it enters the lock phase.
3. **Lock Phase**: The same actions specified in the Fall Phase can be done during this phase as long as the tetromino is not yet locked to the playfield. Hard-dropped tetrominos lock immediately. It enters lock delay before it actually locks. It then enters the Pattern Phase after it fully locks.
4. **Pattern Phase**: The engine looks for any patterns made by the four individual blocks locked down, most often checking for full rows of blocks on the playfield.
5. **Iterate Phase**: Used to allow the engine to scan through all the cells in the playfield to evaluate or manipulate them if necessary. This phase is not often looked at, and used mainly for more complex future Tetris variants.
6. **Animate Phase**: All animations on the playfield are executed during this phase (e.g. animation for clearing lines). It then moves on to the Eliminate Phase.
7. **Eliminate Phase**: Any square blocks that are marked for removal (i.e, blocks in a full row) are cleared from the playfield. If this results in one or more empty rows of blocks, then all blocks above the empty row(s) fall down, or fall by the number of cleared rows. Points are then awarded to the player according to the Tetris Scoring System.



8. **Completion Phase**: Any updates to game information shown on the game interface are updated, such as the score and time elapsed (though score should also be updated when soft dropping, and time should be updated every frame). The level up condition is checked to see whether the game level should increase, and then control loops back into the Generation Phase of the next tetromino. There is a 0.2 second delay during transition from the Complete Phase to the Generation Phase.

lands on a surface, it waits for 0.5 seconds until it locks. There are three settings for lock delay:

- **Extended Placement Lock Down:** The lock delay can be reset before the 0.5 second period ends by rotation or horizontal movement up to 15 times.
- **Infinite Placement Lock Down:** Rotation and horizontal movement can reset as many times as you want.
- **Classic Lock Down:** The delay cannot be reset by rotation or horizontal movement.

Regardless of the setting, the delay will reset if the tetromino falls further down the playfield.

Comparison of Tetris versions researched

Version	Game Boy (1989)	NES (1989)	DS (2006)	Official Website (2014)	Standard (Guideline) (2001 - present)
Playfield dimensions (wxh)	10x18	10x20	10x43 (rows above 20 are hidden)	10x20 visible, 7 extra rows hidden	10x22 (rows above 20 are hidden)
Number of next pieces	1	1	6	3	1-6 , though not required.
Next queue location	Bottom right	Mid-right	Top-right	Top-right	Top right of the playfield
Hold piece function	No	No	Yes (at top-left)	Yes (at top-left)	Yes (required, should be at top-left)
Hard drop (instant fall)	No	No	Yes	Yes	Yes (required)
Colour coded Pieces	No (in black and white)	No, all types have same colour scheme which changes as the game level increases	Yes (follows guideline)	Yes (follows guideline)	Each tetromino type is assigned a fixed colour.
Rotation System	Nintendo Rotation System (left hand sided) *	Nintendo Rotation System (right hand sided) *	SRS	SRS	Super Rotation System (SRS)
Ghost piece function	No	No	Yes	Yes	Yes (required)
Uses standard Random Generator?	No	No	Yes	Yes	Required
Scoring System	Original Nintendo Scoring system *	Original Nintendo Scoring system *	Uses guideline scoring system	Uses guideline scoring system	Defined Standard Scoring system

Requirements

The requirements and success criteria for my solution will be stated in this section, giving a reason for why each exists. These requirements are heavily based on the essential features and my approach

A) Functional Requirements

These requirements involve the features that the game must have, as well as the components which must be displayed on the game interface. Most of these features are present from the guideline and heavily based on the essential features and my approach I have specified earlier based on the research.

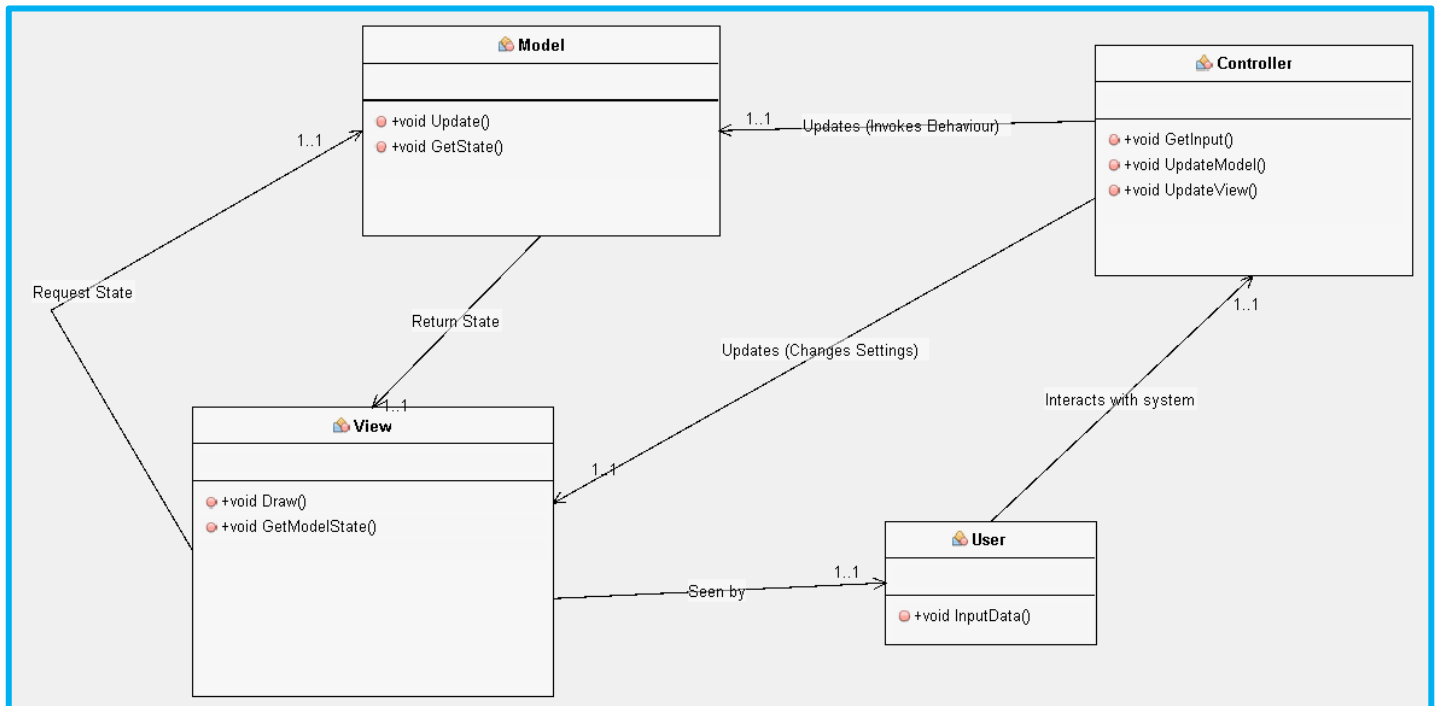
The requirements can be split into three parts: The actual design of the game – what the user will see, and the actual processing of the game – what the program code must carry out, and the input requirements – what inputs that the game must allow.

1) Design and Output Requirements

These requirements are concerned with what needs to be displayed to the user.

They can often be measured simply by checking that it is present.

#	Requirement	Rationale – why does it exist	Success Criteria
1.	The game interface must use a red background	Emphasises the school's house style, where red is the main colour	<ul style="list-style-type: none"> The background of the program window has a red colour.
2.	All text on the screen must be white	Bright colour makes it easy to read.	<ul style="list-style-type: none"> The text displayed on the screen is white. (Can simply be measured visually)
3.	The background colour any text is displayed on must be black	Since text is white, using a black background contrasts well and ensures the text is clear	<ul style="list-style-type: none"> All boxed containers on the screen holding text are filled with a solid black background.
4.	The outline of any boxed data containers is white.	Contrasts well with the black background they have.	<ul style="list-style-type: none"> All boxed containers on the screen holding text are given a white outline.
5.	Has a grid with a width of 10 units and height of 22 units	Clearly specified in Tetris Guideline (standard dimensions)	<ul style="list-style-type: none"> 10 square units of blocks can fit on a single row. 22 square units of blocks can stack up
6.	Grid cells above the 20 th row must be hidden.	Standard in Tetris games (21 st and 22 nd rows hidden)	<ul style="list-style-type: none"> The player can only see the bottom 20 rows of the grid.
7.	The grid must be clearly visible	It visually represents the gameplay	<ul style="list-style-type: none"> The grid is displayed on the interface It will be the largest component shown



A UML diagram used to show how the model-view controller pattern works

Design of User Interface

The view component of the program will be designed first to allow the client and potential users to get a 'look and feel' of the user interface of the program before progressing further and developing the model component. No functionality of the program will be designed or developed at this stage. **The only screen being designed and developed at this stage is the actual gameplay screen, not a 'start screen', 'pause screen' or a 'menu screen', since they are not as important.** Therefore, when the program is run, the game interface screen will be shown immediately along with all of its components.

Interface

The initial design and layout of the interface will be provided to show to the client, who will be able to make amendments before it is used for developing the game interface of the program.

Input

The program will be using various forms of data input. Most user input will be entered from the keyboard, used for the controls of the Tetris-like game. The user input was already indicated on the screen design for the game interface.

Input is also randomly generated by the program itself, specifically the order of which different types of tetromino pieces (Tetris shapes) are generated on the playfield.

In the case that high scores are used, the program will read data from a flat file which stores the high scores, to display the highest score while the game is being

The pseudocode below shows the basic implementation for the main loop of the program, provided by the **MainLoop()** method of the GameController class.

The **get_user_input()** and **delay_time()** methods already have some form of implementation by Pygame.

In the pseudocode, the line for checking for when the exit button is pressed is written in words rather than code, since I cannot simply implement this; Pygame would provide a method to carry this out with actual code.

```
public procedure main_loop()

    // record whether the game should end.
    ended = false

    // The actual main loop.
    while NOT ended

        input = get_user_input() // get a list of all user input.

        // end game if the exit of the display window is pressed.
        if exit button within input then
            ended = true
        endif

        // update the model
        update_model()

        // update the display screen of the program.
        update_view()

        // Delay the program for a very short period of time to
        // limit the number of frames per second.
        delay_time()

    endwhile

endprocedure
```


First Iteration - Development

Day #1 -- Date: 08/12/17

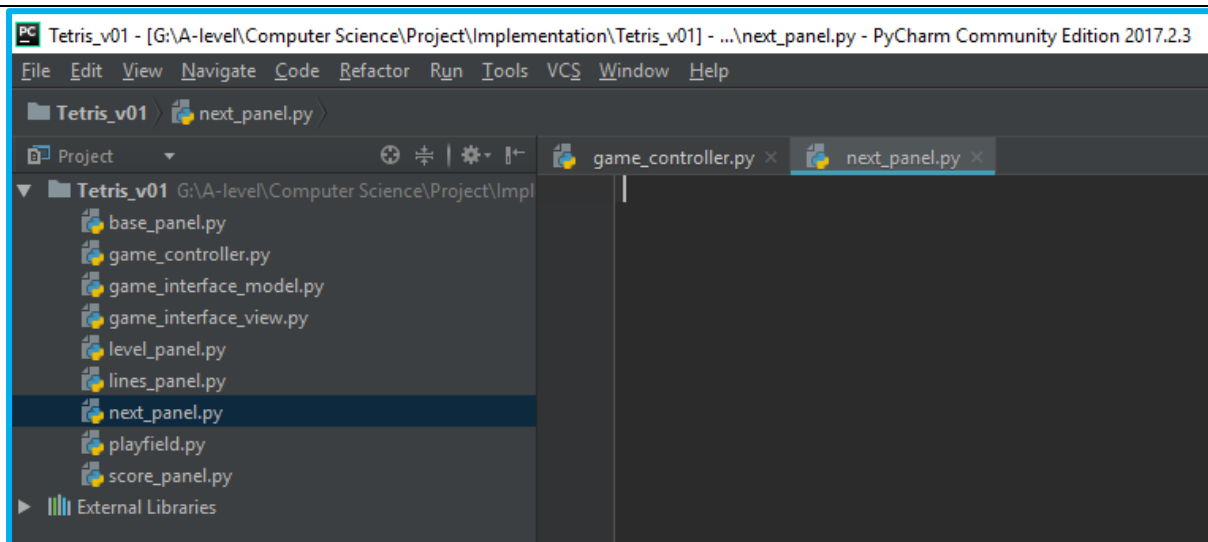
I have now begun to implement my first design for the project, using the Python programming language assisted with the Pygame module.

Choosing an IDE

There are a variety of possible IDEs that I could have used to help with the development of my program, such as IDLE which is bundled with the Python installation. However, IDLE is rather basic for developing a program. The IDE which I found most suitable for the development of this program is the PyCharm. It is made specifically for the Python language, and supports many features such as code completion, syntax and error highlighting and code refactoring (renaming and restructuring code without altering its functionality). It also provides assistance for documentation of code through auto-completion to set up the structure. I used the free Community Edition of PyCharm to develop the program.

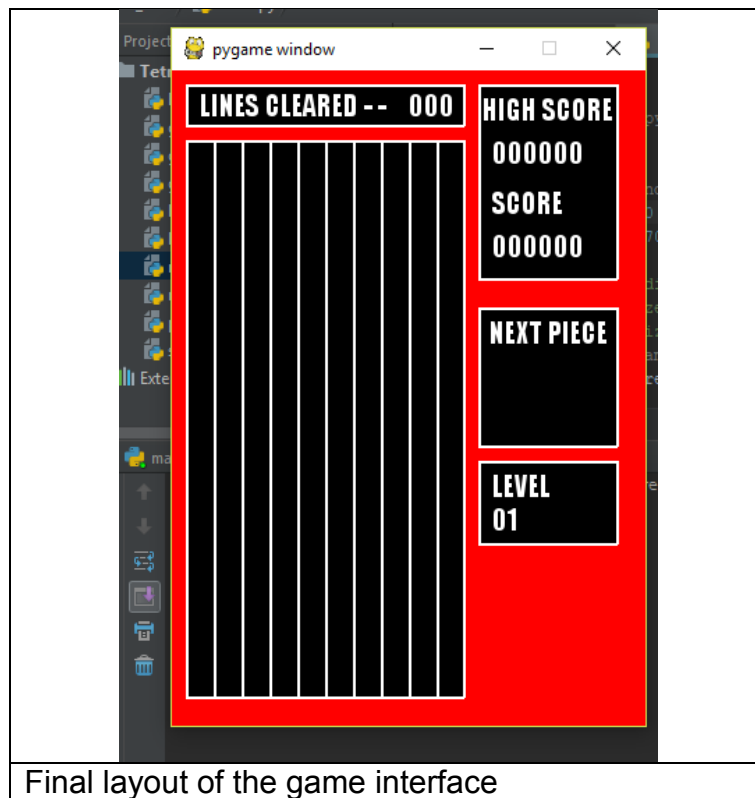
Setting up PyCharm project

To begin with, I created a new project in the PyCharm IDE in a folder called 'Tetris_v01'. Next was to create a python file for each individual class from my design. The reason for creating them in separate files was to keep a modular approach, and to ensure the program has a good structure to enable it to be read and modified easily. I gave each file the same name as their respective class, but in lower-case, and each word separated by underscores. This is a standard naming convention used for program source files not only for Python, but many other programming languages. After all of said actions are done, the project structure and file names should be in the form seen in the image below.



Project structure created in PyCharm.

The layout was improved even further by decreasing the width of the screen by 60 pixels, and decreasing the height by 30 pixels. The **rect-x** coordinate of the ScorePanel, NextPanel and LevelPanel classes was decreased by 30. As a result, the layout was made far better, making more use of space, and leaving a gap for where the pause button will be displayed in a later iteration.



Second Iteration: Design, Test Strategy, Prototyping and Development of User Interface, Input and Output Component

Start Date: 21/11/2017

End Date: 02/01/2018

By the end of this iteration I need to make sure that the following occurs in the program:

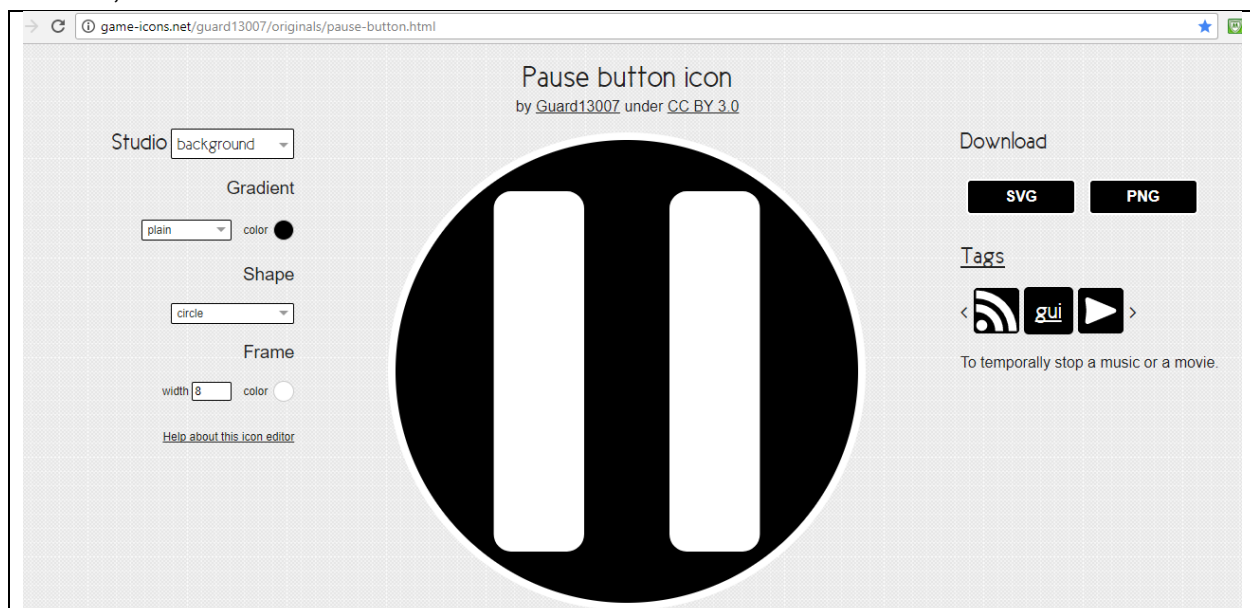
- An image of the next tetromino is displayed correctly within the next piece panel
- The pause button is displayed correctly
- The horizontal gridlines of the playfield are shown
- The program is positioned to the centre of the screen

To meet these points, I first need to design the format for each tetromino piece being displayed, and also provide a design for the pause button.

The design of the pause button

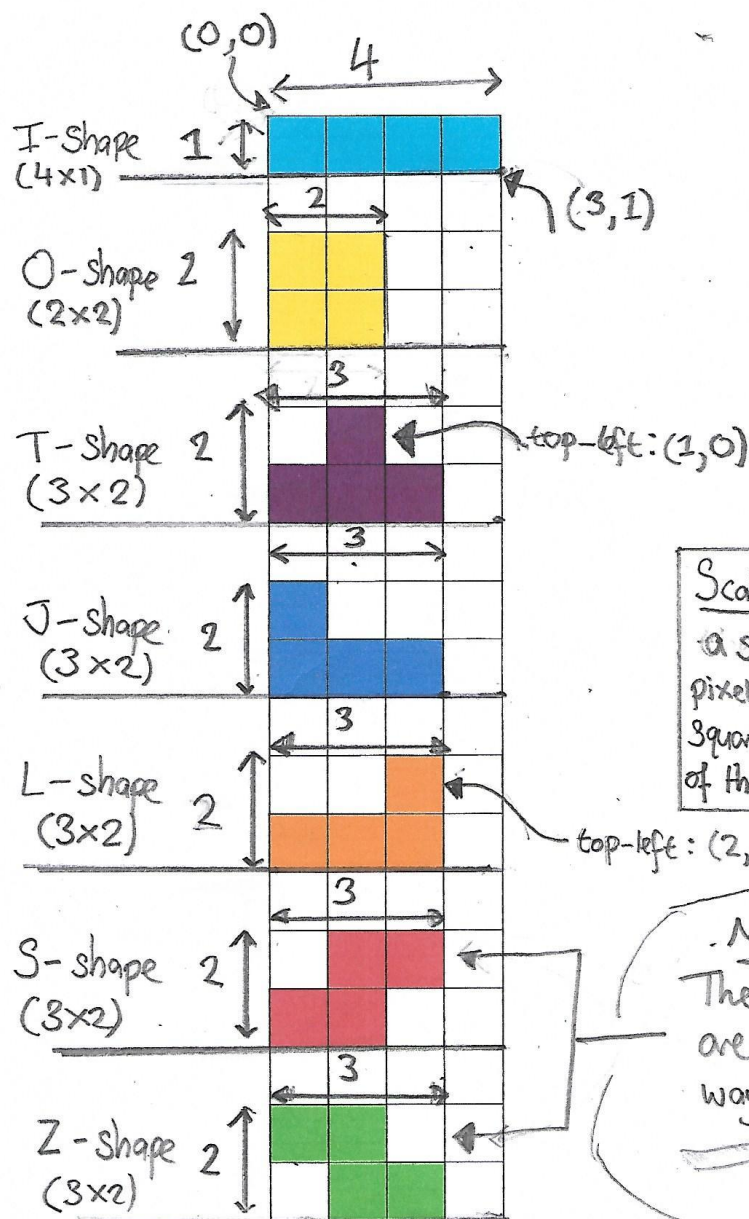
Instead of using the draw module provided by Pygame for creating computer graphics, I have decided to create an image of the pause button which will be imported into the program. This is because the pause symbol will be complicated to draw from scratch, and circles drawn using computer graphics often have unsmooth edges which can make the button seem uncomfortable to look at.

I came across the website called **game-icons.net** which provides customisable game related graphics of many different categories. It helped me to design the pause button, as shown below.



Pause button designed using game-icons.net. Has the format as described in requirements specification.

Tetromino Image Layout



Scale: 1 unit has
a square side, in
pixels, equal to the
square scale
of the Playfield

Mistake:
The colours
are the wrong
way round.

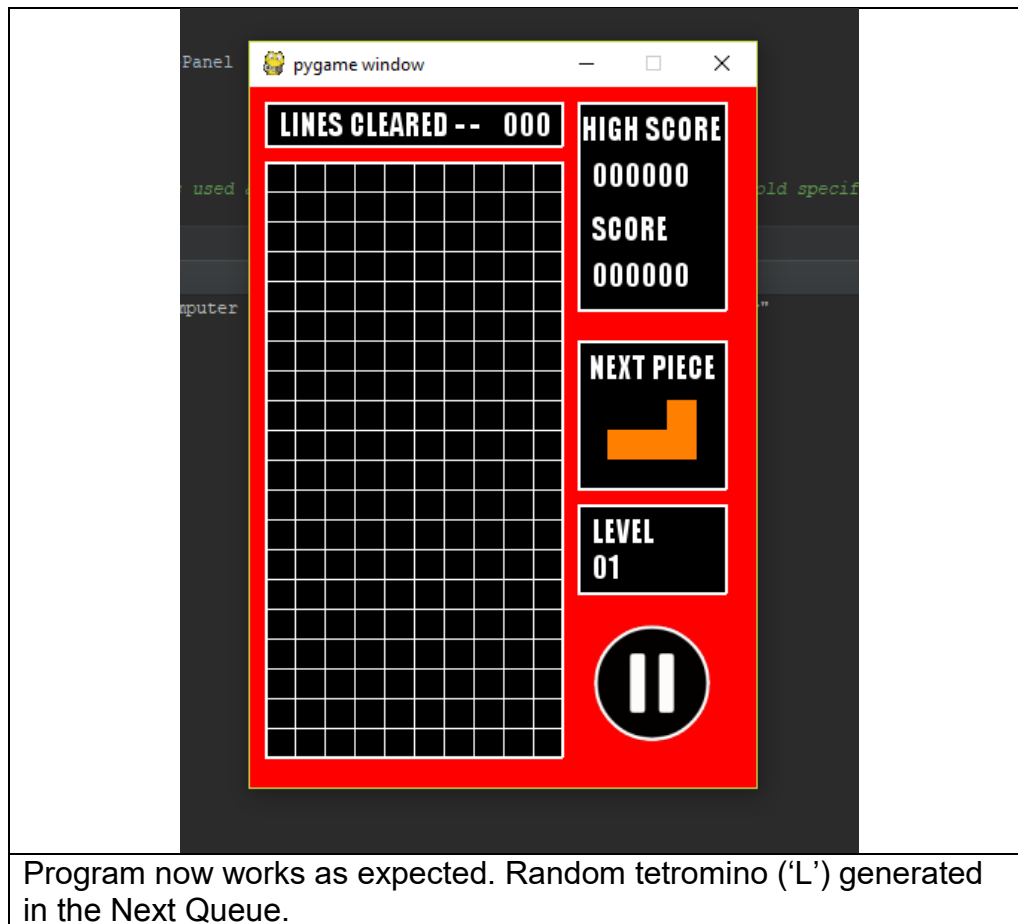
Client Signature

Sarah

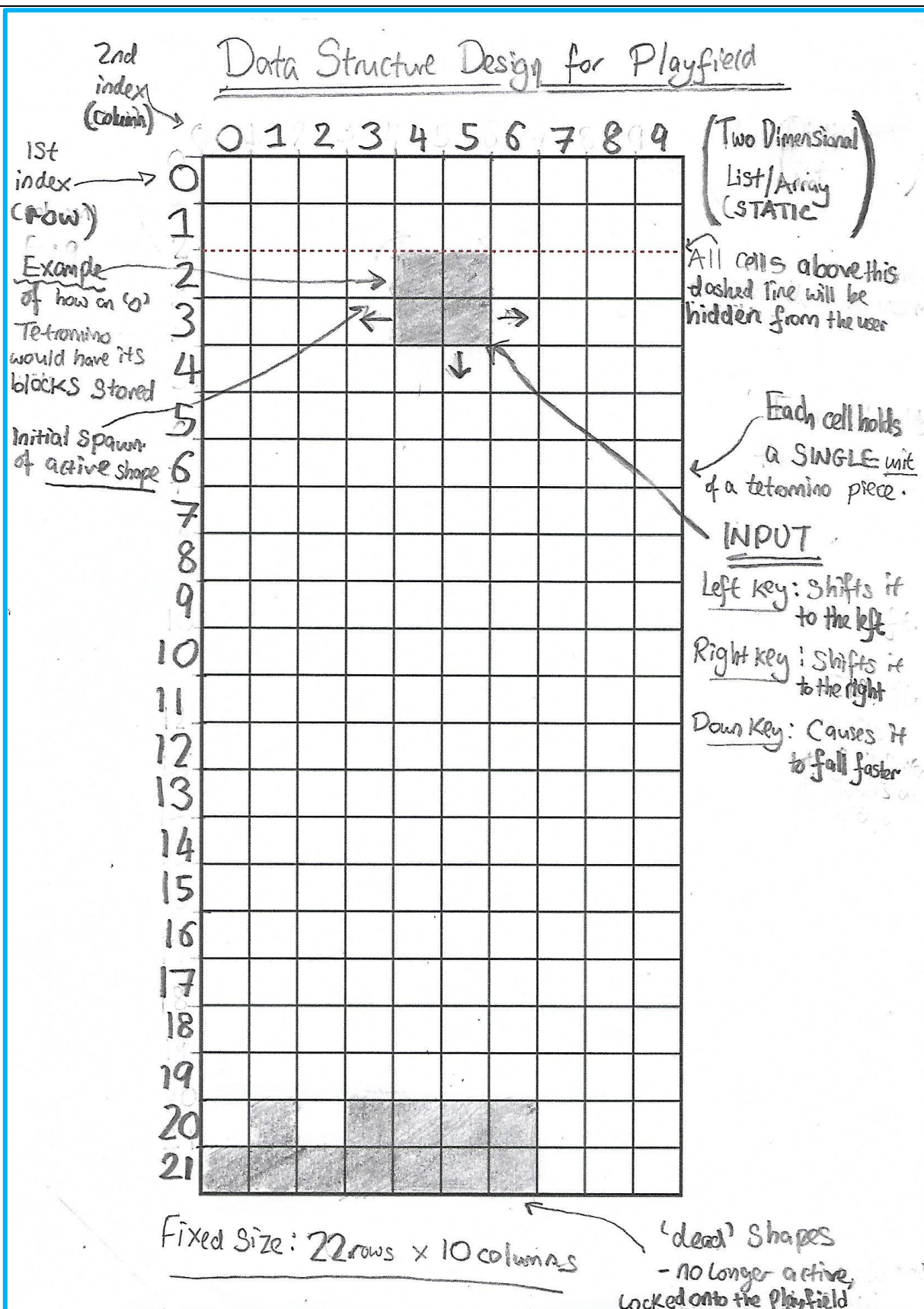
Tetromino Layout Design

Running the program after fixing the error

I ran the program again to make sure that it now works after attempting to fix the error. As shown in the screenshot below, the program works again.



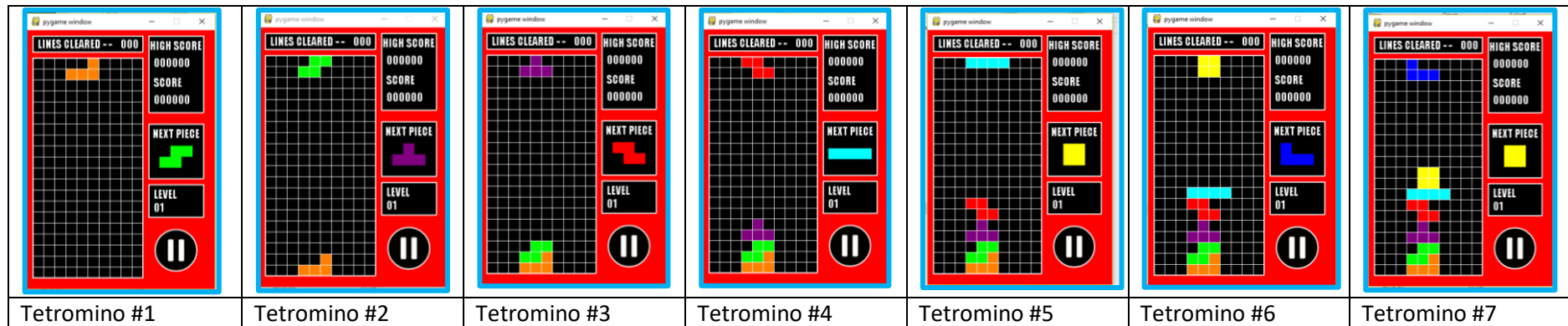
Here marks the end of the development of the program for this iteration.



Screenshot of the playfield data structure design.

Results of Test #5, 6 and 7

The remaining tests can all be performed together, since they all involve checking how tetrominos spawn on the playfield. The screenshots below sum up all three of the tests.



- **Test #5** was successful, because as shown in the screenshots, all new tetrominos that spawn on the playfield match the tetromino displayed in the Next Queue before it spawned. Each different tetromino's colour, structure and orientation match exactly as shown by their respective image in the Next Queue.
- **Test #6** was successful, as the spawn locations of each tetromino piece are correct. Each one spawns at the top visible row, and the middle (or middle-left) column as expected.
- **Test #7** was also successful. As clearly indicated by the screenshots, the tetrominos spawn as if they are pulled out randomly from a bag without being replaced until the bag is empty. The first 7 tetrominos that spawned consist of each possible type. It is implied that the sequence order is random, because in the screenshot showing tetromino #7, the tetromino that will spawn next (Tetromino-O) does not match the first tetromino that spawned (Tetromino-L).

Result: All three successful.

Extra Evidence: Test#7: [page 235](#) (shows another sequence of spawned tetrominos)

Fourth Iteration: Design, Test Strategy, Prototyping and Development of User Interface, Input and Output Component

Start Date: 05/02/2018

End Date: 20/02/2018

By the end of this iteration, I want the following to be implemented in the Tetris program:

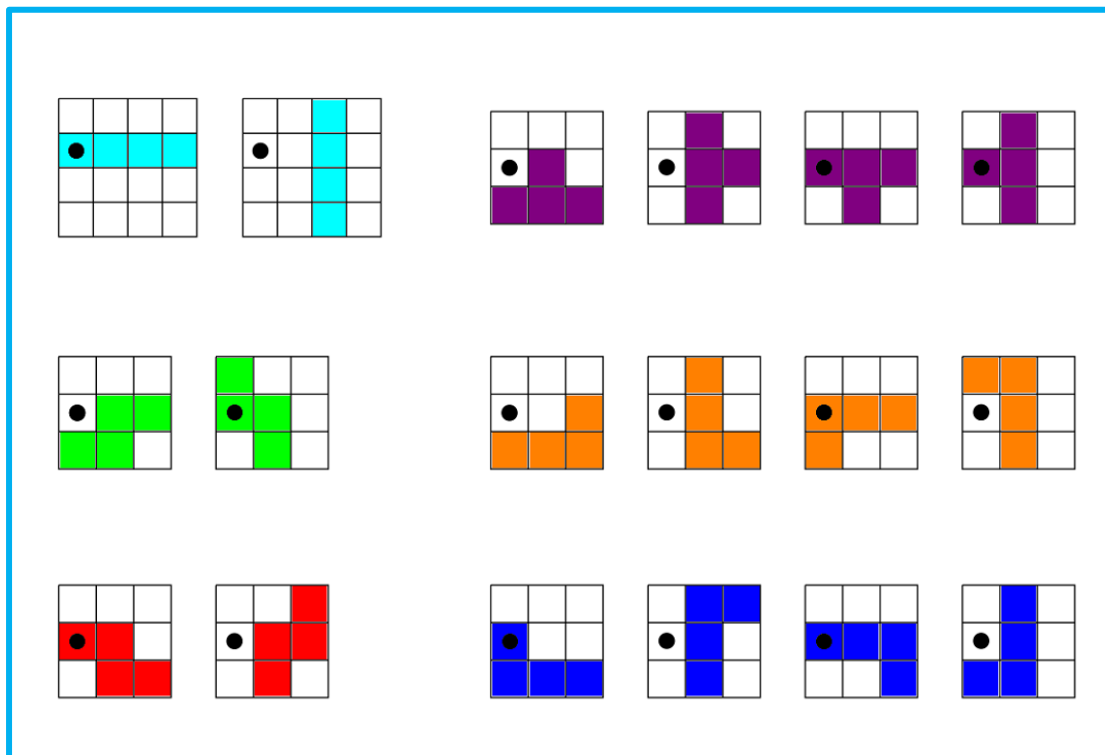
- Implementation of the rotation system.
- Line clear functionality
- Top out (getting a game over)

Design for the Rotation System

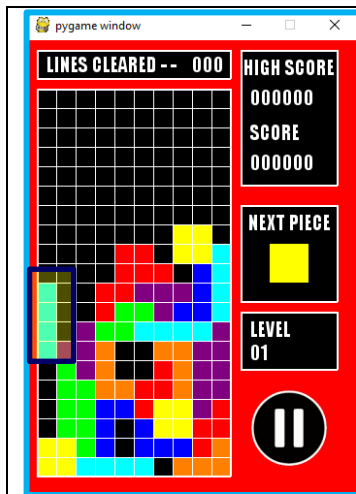
With the combination of DrawPlus X6, GIMP and Microsoft Paint, I created designs for how each tetromino piece will be rotated. Tetromino-O is an exception, as it has a square structure meaning that it will remain the same if it were rotated.

For each tetromino, the orientations are arranged from left to right in a sequence as if the tetromino was rotated clockwise at 90 degree intervals, with the last orientation cycling back to the first.

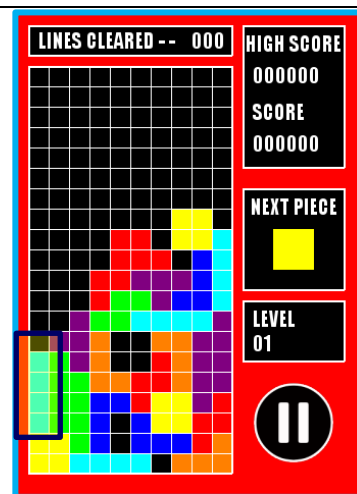
Duplicate orientations are ignored, hence why the 'I', 'S' and 'Z' tetrominos are only shown with two different orientations.



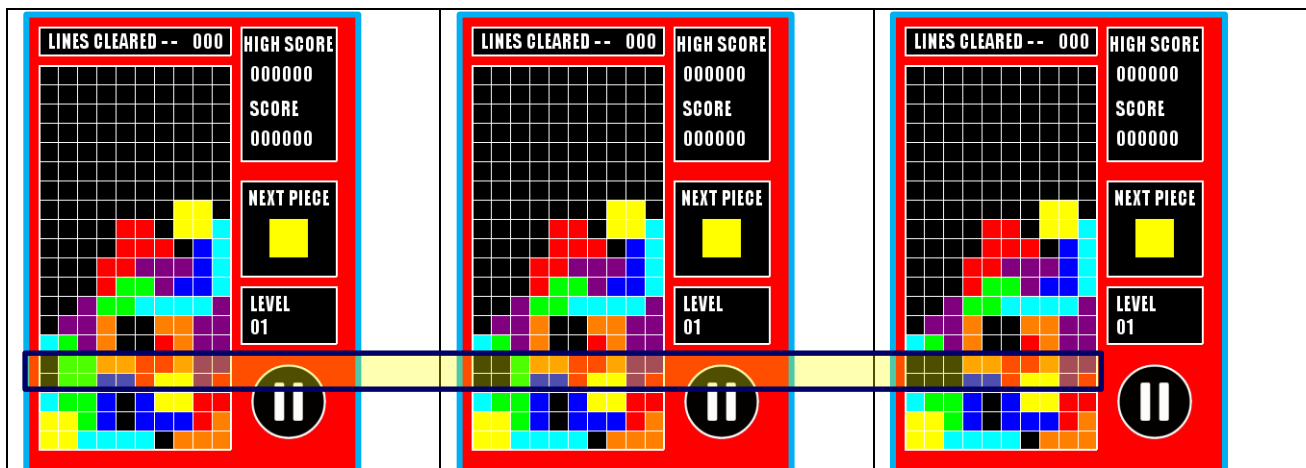
The program was executed again to test the line clear functionality once more. The logical error was no longer present, and the lines were cleared correctly with the animation working properly.



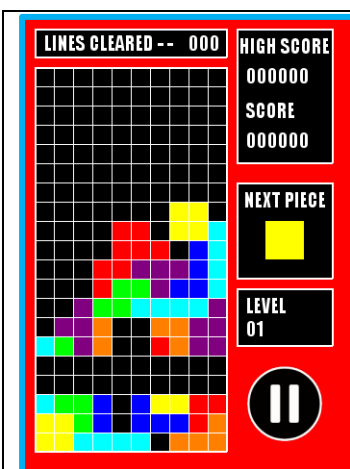
1) Tetromino-I on the left column is the active tetromino.



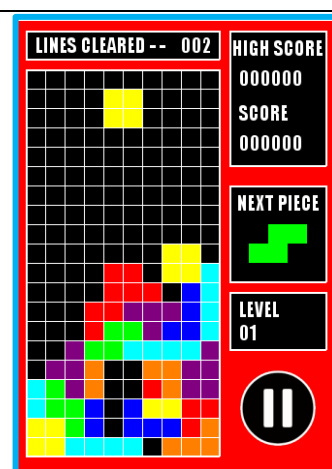
2) Tetromino-I is locked at the left column. Two lines are to be cleared.



3) The clearing animation plays



4) The rows are cleared, and the animation ends.



5) The rows above the cleared lines are shifted down. The value for lines cleared increases by 2, and the next tetromino spawns.

Fifth Iteration: Design, Test Strategy, Prototyping and Development of User Interface, Input and Output Component

Start Date: 02/03/2018

End Date: 15/03/2018

It is very likely that the development of the Tetris program will end after this iteration, due to **time constraints**. The following needs to be accomplished by the end of this iteration:

- Pause button functionality
- Storage of high scores

High Score permanent storage

I have decided to store the high scores in a JSON (JavaScript Object Notation) file, which be given he following filename:

“high-scores.json”

It will be restricted to only store the highest 5 scores to prevent it from becoming overly large.

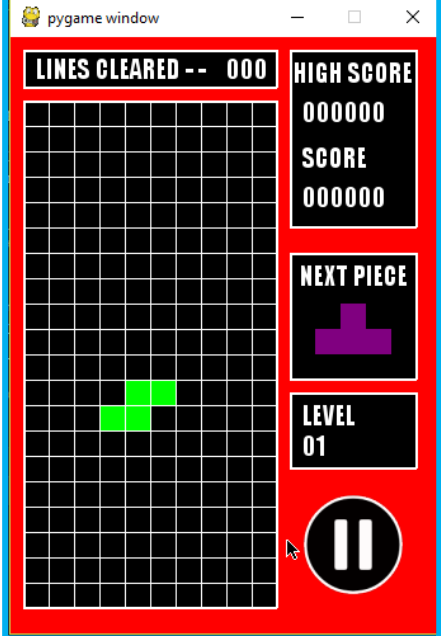
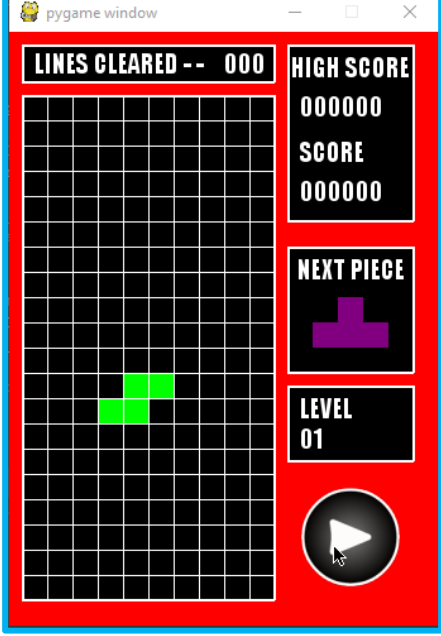
The JSON file structure involves holding attribute-value pairs to store data. In this case, the only attribute stored is given by the name **“high scores”**. It is mapped to an array used to hold the value of each high score.

In order to match the requirements specification, it will initially be filled with zeros. This means that when the user first plays the game, the highest score to beat will be set to 0.

```
{  
  "high scores":  
    [  
      0,  
      0,  
      0,  
      0,  
      0]  
}
```

Testing the pause functionality

The pause button was tested to make sure that it can be used to pause/un-pause the game. After clicking the pause button once, the game stops, and the active tetromino stops falling. After it is clicked again, the game resumes from the same point. This was also tested for the escape key, and it worked the same way.

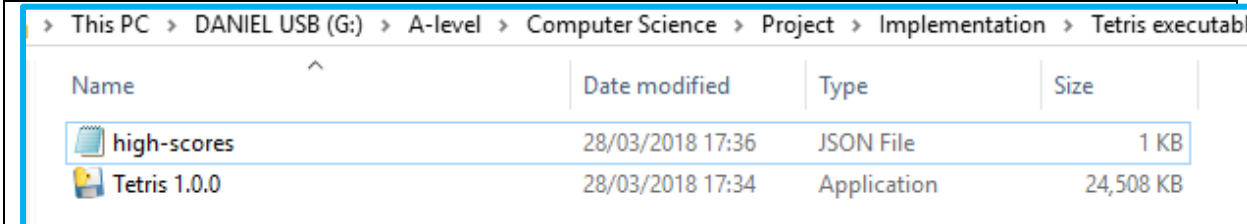
	
<p>1) Before the pause button is clicked on. The game is being played as normal.</p>	<p>2) After the pause button is clicked on, the game pauses. The pause button image becomes a 'resume' image.</p> <p>It can be seen that the mouse pointer is hovered over the pause button, and hence the 'glowing' image is shown.</p>

Evaluation

Executable file

Through the use of **PyInstaller**, the developed program was bundled into a single executable file as shown below. Before doing so, I made sure that the values stored in the high-scores JSON file were set to 0. Since the data within the high-scores JSON file will not remain fixed, it was not included in the bundle. These files were placed inside a single folder. The executable file has been tested to make sure it worked correctly.

When placed within a compressed (ZIP) folder, the overall size was approximately 24 MB. This is significantly larger than the total size of the program files and images.



Name	Date modified	Type	Size
high-scores	28/03/2018 17:36	JSON File	1 KB
Tetris 1.0.0	28/03/2018 17:34	Application	24,508 KB

Testing to inform evaluation

The development of the program has been completed, and so I devised from post-development tests for the functionality of the program, as well as its robustness. Testing for User Acceptance will also be done, using a questionnaire to gather the results.

Functionality

Only two tests were necessary for testing the functionality of the system.

The first test involved checking for any code that is redundant, and never executed at all – ‘dead code’. The **PyCharm** IDE came in handy to identify such code, as it has a feature specifically for this task.

The majority of the code was not redundant, with each class having all of its attributes and most of its methods used at some point. There were, however, a few unused import statements. Such lines were therefore taken out.

The table below shows the methods that were never executed. Each of these methods were removed from their class.

Class	Method
LevelPanel	set_level()
LinesPanel	set_lines_cleared(), get_lines_cleared()

Questionnaire Results*Questions with Yes/No response:*

#	Question	Yes %	No %
1	Have you ever played Tetris before?	60%	40%
11	Would you agree that the shapes are generated on the screen randomly?	100%	0%
12	Do the shapes automatically fall faster as the level of the game increases?	100%	0%
13	Do the shapes fall faster while the down arrow key is pressed?	100%	0%

Questions with 1-10 Scale:

#	Question	Results	Average
2	How clear (well laid out) do you find the layout of the game interface?	10, 9, 8, 7, 10	8.8
4	How clear to read is all text and images displayed on the screen?	10, 9, 10, 9, 10	9.6
7	How comfortable are the controls of the game?	8, 7, 7, 9, 9	8
9	How well does the program work in terms of user-friendliness?	6, 7, 5, 7, 7	6.4
15	How much do you enjoy playing the Tetris game?	8, 9, 8, 5, 8	7.6
16	How much does the game motivate you?	7, 7, 9, 7, 8	7.6

Questions with multiple options (ticking only one)

#	Question	Results
5	What is your preference for the size of the game screen?	Smaller screen: 0% Current screen size: 40% Larger screen: 20% Full screen: 40%

Client Written Review

When playing the final version of the game, I was impressed with how well it matched the gameplay found in many variations of Tetris. The design of the interface was done professionally, with all the displayed game information is laid out flawlessly, and a well-suited colour scheme. The controls of the game are comfortable and easy to use, though would be much appreciated if it mentioned them to the user.

I do wish for a more music and sound effects were added to make the game 'flashier' and help to keep the students engaged, but of course this is of low priority. I acknowledged the inclusion of the block clearing animations played when scoring lines and getting a game over to help deal with this.

I have watched the students testing the game numerous times, and noticed their gradual improvement and attachment to the game. I gave them a mental maths test one week after the game was distributed for testing in order to assess how well they have improved. They did slightly better than before and seemed more focused.

Overall it is a very simple but well-designed game that will keep my students trying to improve their high score. I believe it will help my students to think more logically and improve their critical thinking skills when solving mathematical problems, especially those relating to mental maths. Though it would be even better if additional features were added to make the game more educational and more unique compared to other variants, but I am aware that the problem relating to time constraints did not allow this.

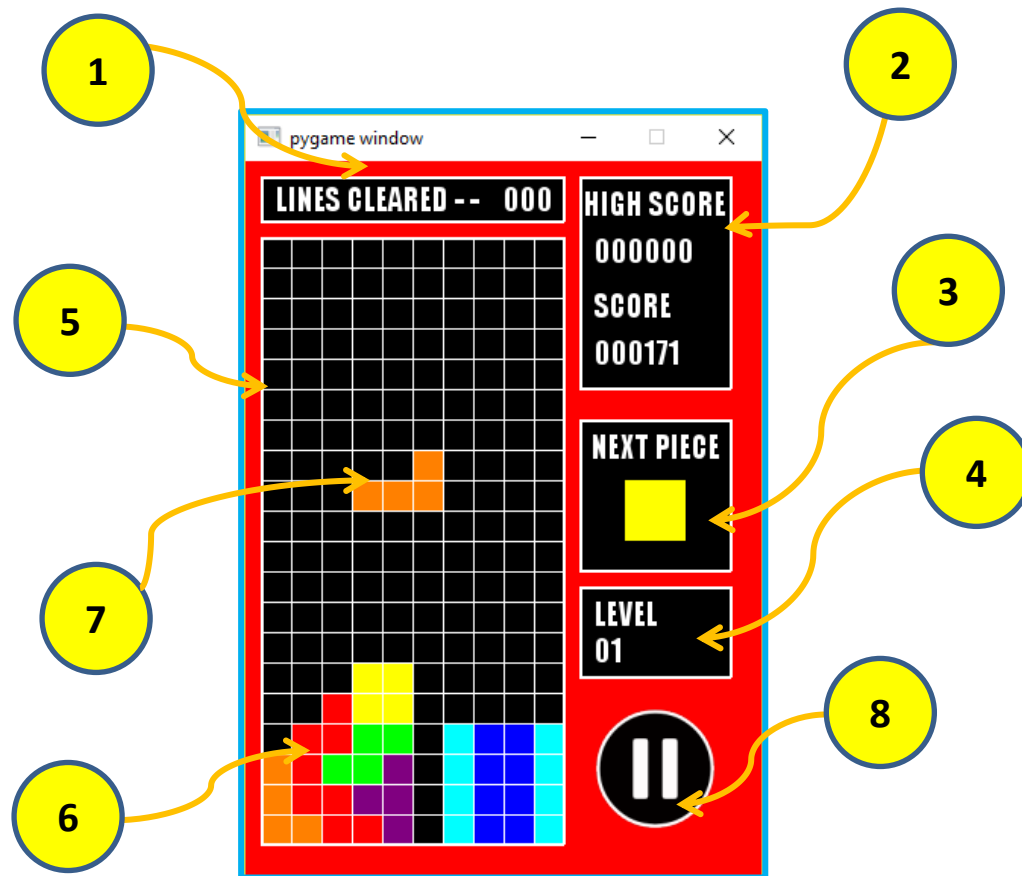
- 1. Mental Maths Skills Rating: 8**
- 2. Spatial Skills Rating: 8**
- 3. Student Progression Rating: 7**
- 4. Perceptual Skills Rating: 7**

From my client's review, it is clear that she was satisfied with the developed system, and that it was able to meet its requirements. She did mention that the game was 'simple' and could be made 'more unique' if educational features were added.

She gave scores out of 10 for how well she felt the student testing group improved their mental maths skills, spatial skills, progression through playing the game and perceptual attention skills.

Description and Usability features of the program

The usability features of my program are very limited due to the lack of navigational features. Limitations #2 and #4 address the main issues regarding the usability features. However, based on the results of the questionnaire, the layout of the game interface has been very well done, as all the game information is displayed clearly in an organised manner.



1. The '**Lines Panel**': Displays the current number of lines cleared by the user.
2. The '**Score Panel**': The number at the top is the current high score the user has obtained, while the bottom is the current score.
3. The '**Next Queue**': Shows the next piece that will spawn on the playfield after the current one being controlled locks down.
4. The '**Level Panel**': Shows the current game level, which determines how fast the active piece will fall, and the number of points awarded.
5. **Playfield**: The grid where the game is played, holding all the square units of the tetromino pieces that spawn on it.

Bibliography

Software

- SerifDraw Plus X6 - Design
- Microsoft Paint - Design
- PyCharm IDE – Development of program
- GIMP 2 - Design

Websites (all hyperlinks checked and accessed of 14/03/2018)

- Hardrop wiki – used for research on Tetris
https://harddrop.com/wiki/Tetris_Wiki
- Tetris wikia – used for research on Tetris
http://tetris.wikia.com/wiki/Tetris_Wiki
- Python documentation – lookup of built-in modules
<https://www.python.org/doc/>
- Pygame documentation – lookup of the Pygame library
<https://www.pygame.org/docs/>
- <https://www.pygame.org/wiki/FrequentlyAskedQuestions> - provided the code to centre the program window.
- Tetris Website – to actually play Tetris for free
<https://tetris.com/play-tetris>
- gameicons.net – resource used to design the pause icon.
<http://game-icons.net/>
- <https://jeffknupp.com/blog/2013/04/07/improve-your-python-yield-and-generators-explained/> - Helped to understand generators
- Pyinstaller – Used to bundle my program into a standalone executable file.
<http://pythonhosted.org/PyInstaller/index.html>

Misc

- 2009 Tetris® Design Guideline – used for research on Tetris, where images shown on pages **7**, **8** and **10** originated from.