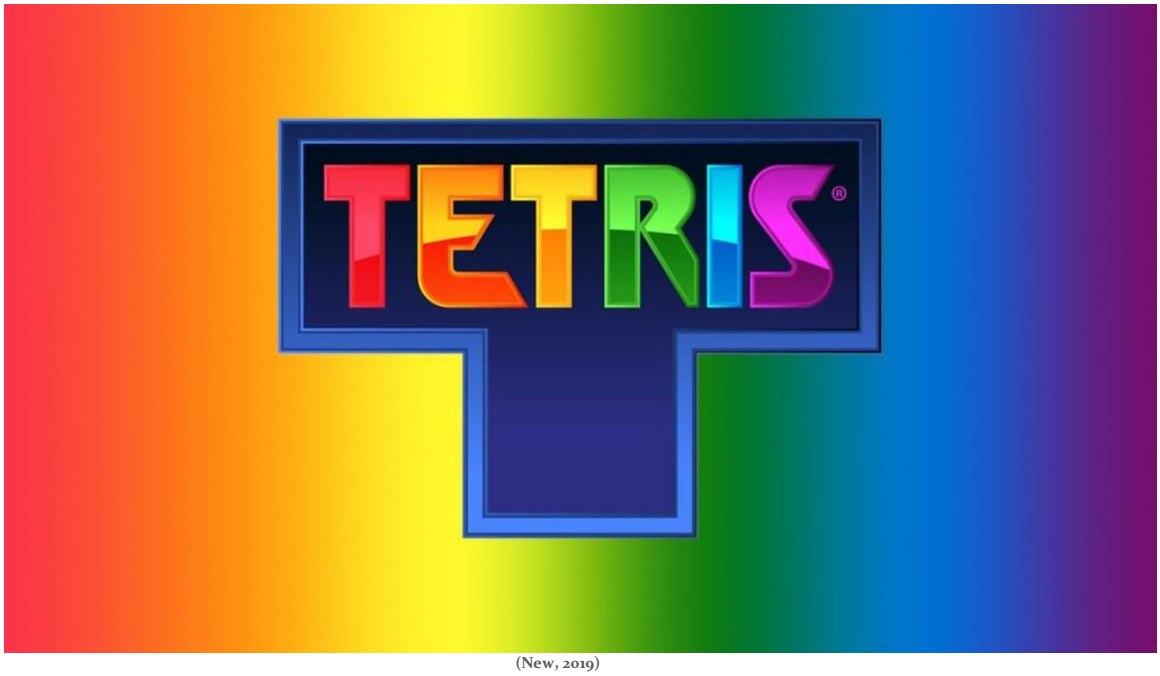

(New, 2019)

Can I successfully code a fully working version of Tetris using Python without any prior knowledge?

BY ARYAN AGRAWAL

Candidate Number: 8011 | British School of Brussels | Centre Number: 74303

Extended Project Qualification
Word count: 6243
Year: 2019

Contents

Introduction

Formation of the Idea

I initially got the idea for my EPQ project when I discovered that artificial intelligences could play computer games; I had just discovered that AlphaGo Zero, an artificial intelligence (AI) made by Google mastered Go, a popular board game (Gibney, 2017). After doing some further research, I found out that people were making machine learning algorithms that could play games like the Chrome Dinosaur Game (AI learns to play Google Chrome Dinosaur Game || Can you beat it??, 2018). With the knowledge that it was possible to create an AI for games, I decided that I wanted to try this concept out for myself on a smaller scale, so I chose to make an AI for a classic retro game of which I considered: Tetris, Minesweeper and chess. I chose these games specifically due to my vested interest in all of them; I had learned to play chess from a young age and was always fascinated by the seemingly infinite number of moves possible in the game. I promptly realized that chess would be too challenging to create an AI for, especially at my level of coding experience, as it had taken years for many professional programmers and mathematicians to make an AI for it which could actually beat humans (Wikipedia Contributors, 2019). Minesweeper seemed like a very interesting game, which I enjoyed playing due to its simplicity but difficulty to play as the board got bigger. I initially thought that the game would be too easy to make an AI for due to its simplistic gameplay, and obvious choices for what to do next in the game (from experience), so I selected Tetris, which I decided originally to be a robust middle ground in terms of difficulty, not being too out of reach for a beginner programmer but not as "easy" as Minesweeper to program an AI for. I did not know this for sure, as I only had the experience of playing these games, and I noticed the strategy used in Tetris was far more complex than that to complete a game of Minesweeper. Without any experience in coding the games, I made the false assumption that making a version of Minesweeper from scratch, including learning to actually code, would be an easy feat.

I had also considered solving one of the millennium prize problems after I briefly heard of them but finding out that the greatest mathematicians could not solve them for years using the most advanced maths (Clay Mathematics Institute of Cambridge, Massachusetts, 2013), I chose to go with the artificial intelligence idea.

Almost halfway through the EPQ, I decided to slightly change my project: initially, I had planned to make my own version of Tetris, and then code an AI for it (as I would have a greater understanding of how the game worked if I made it myself), but I soon realized that even this prospect would be far too ambitious for the coding beginner. I settled on making just the game, as that would not only let me develop my foundation in Python, but also allow me to put all my effort and focus into one main project, rather than worrying constantly about another component of the project that would be even harder to make. Due to the way that I had coded my project initially, I would have to restart the making of my project on a clean slate, bringing with me fewer mistakes, better coding skills and more experience.

Goals & Ambitions

The main aim of the project was to create a version of Tetris accompanied with an artificial intelligence that can play the game to a proficient level, beating the average human player. I also intended to improve my coding skills with different types of projects compared to what I have done in the past – I have had experience with VB.NET, making mostly console based programs and a very simple calculator program. Embarking on this project would be a totally different realm of computer science and programming for me, which would help me broaden the scope of my coding knowledge. Furthermore, I wanted to gain the experience of coding in a language like Python, which is commonly used in various commercial applications; Python is used for special effects in movies as well as at Google, Facebook, Instagram, Reddit, Spotify, Quora and Netflix just to name a few (Real Python, 2018). Additionally, I wanted to gain the knowledge of the process of creating a project from scratch, and to see if this is something I would be interested to do in the future, such as in university or future careers. I also aimed to learn how to problem solve independently and to learn how to overcome the unexpected obstacles frequently experienced when making a game (MakeUseOf, 2017), or any new project for the first time.

Proposed Course of Action

Tetris Game Research

Playfield: a **10x20 grid** will be used (to allow for the players to have enough time at the beginning of the game to see the pieces without having the need to hide part of the playfield)

Spawning: all the pieces will **spawn in the middle**

Tetrominos: these will be the same as the original game, including shapes and colours (**I (cyan), O (yellow), T (purple), S (green), Z (red), J (blue), L (orange)**)

Rotation: the **SRS (Super Rotation System)** will be used to rotate the pieces correctly according to the guidelines, in order to keep the game mechanics as similar to the original game as possible

Controls (when human is playing): ***WASD*** keys will be used to control the Tetrominos, where *W* drops the piece instantly (hard drop), *A* moves the piece a unit left, *S* increases the speed of the piece dropping, and *D* moves the piece a unit right; also, the ***RIGHT ARROW*** key will be used to rotate the Tetromino clockwise and the ***LEFT ARROW*** key will be used to rotate the Tetromino anticlockwise

Piece generation: a **random number generator** will be used to generate pieces (each will be assigned a number, with equal chances to spawn each piece (as stated in the guidelines)

Scoring system: a basic scoring system will be implemented; **one point will be awarded where for each row that is cleared** (by completing the row)

Levelling up: **for every point scored, the speed of the pieces falling will be increased by a factor of 1.05**; there will be no cap for this increase, so the game will get exponentially hard

Endgame: the **player will lose if there is not enough space to spawn a new piece** (touches the top of the grid)

***Sound effects**: these will not be implemented, as this will complicate the program massively and make the coding of the game less efficient

***Pausing**: you will not be able to pause the game

***Ghost pieces**: these will not be implemented either, as this also complicates the code without adding much value; to compensate for this, the grid will be slightly highlighted to act as a guideline to the players to indicate where the piece will drop

***Next piece preview**: this will not be integrated into the game as it complicates the GUI of the game, which is already very complicated to code

***Saving pieces**: this will also not be integrated into the game, again for the sake of simplicity in the game

Features that won't be implemented:

- Sounds
- Pausing
- Ghost pieces
- Next piece preview
- Saving pieces

The reason these features won't be implemented into the game is to reduce the error that could arise while coding the game; the purpose of the game is not to integrate every feature for the human player, rather, it is to provide an efficient platform for the neural network to play the game.

Fig. A – research document outlining features to implement/exclude

My initial proposed course of action to complete my project was to research and take the best free online courses for coding in Python, to improve my coding skills and obtain basic knowledge about the language, such as: the syntax (structure of code) or common modules (premade functions used for specific tasks). I would use these to carry out tasks such as randomizing, or how to declare functions and variables, etc. I planned to simultaneously research every detail about the game such as the falling speed of the

blocks, the size of the board, etc. which I would outline in a document (Fig. A). I would follow this document while coding the project, in order to know what to code and how; this whole process was meant to take just 1 week, from 04/02/2019 to 10/02/2019 (Agrawal, 2019). I planned to then try and make the graphical user interface of my game, which should have taken the next 2 weeks from 11/02/2019 to 24/02/2019 (Agrawal, 2019); I initially planned to do this using either TKinter or Pygame but chose Pygame due to the fact that I was programming a game so I would need to use Pygame anyways. Afterwards, I intended to code the board, the pieces and their movement on the board, from 25/02/2019 to 03/03/2019 (Agrawal, 2019). Coding the features of the game in Fig. A would come next, and I expected it to take around 2 weeks, from 04/03/2019 to 17/03/2019 (Agrawal, 2019); I wanted to analyse other people's code but didn't end up doing that until the second attempt of the code. I planned to fix all the errors that would arise while coding the project as they came up, as these would be unplanned errors such as syntax errors, but I did not allocate any extra time to fixing errors or improving the game substantially as it would be an ongoing process.

Next, I planned to research the coding of neural networks and artificial intelligences for similar applications such as other 2-dimensional games, to assist me with the final coding of the neural network that would play the game. The research and making of the artificial intelligence were planned to take 4 weeks, split before and after the Y12 end of year exams (18/03/2019 to 31/03/2019 and 03/06/2019 to 16/06/2019 (Agrawal, 2019)) – I planned to do the research before the exams and the programming of the AI after, though I didn't end up going through with both the research and the coding of the AI due to the modifications I ended up making to the project, of removing the AI component of the project and focusing solely on the game after the mid project review (Agrawal, 2019a). The Gantt chart with dates is shown in Fig. 19 and Fig. 20 in the evaluation.

Research

Primary Research

From my primary research, I found out which language I should program in, and what game I should make. I spoke to our computer science teacher about this, and she outlined the benefits and drawbacks of using either Java or Python, the 2 languages I was considering coding the project in. She mentioned that Python is easier to code in for beginners, but Java is more commonly used everywhere and more popular; I concluded that Python would be the better option for my project. She also agreed with my idea that Tetris would be an interesting but demanding game to make due to it being not too complicated but not too simple either. Additionally, I downloaded the game on my phone using the Play Store (ELECTRONIC ARTS, 2009) and played the game in order to get a feel for the mechanics and how the game generally works when it is played. This research also gave me some initial ideas for how to make the graphical user interface (GUI) of the project.

Secondary Research

For my secondary research, I did some further digging into the benefits and drawbacks of Python; by results matched with the information from the computer science teacher (Christopher Castiglione, 2019), (Deven Joshi, 2018). I also learned the basic syntax and coding techniques used in Python, by looking at source code for other projects and taking online bootcamps and coding tutorials; the courses by SoloLearn (SoloLearn, 2019) and edX (Edx, 2018) were useful to a certain extent as I relearned the fundamentals on how to code, but the most helpful was the LearnPython website (LearnPython, n.d.), which goes through the fundamentals as well as some of the advanced skills used in programming specifically with Python. On these websites, I answered quizzes throughout the duration of the course to ensure that my progress in learning Python was sufficient. I did not complete the courses fully though, as I thought that the very advanced coding concepts would not need to be used in the making of the project, meaning I could use that time on improving my own program; I could always come back to the courses in necessary.

In addition to this, the source code provided by the sites: Rosettacode (2017), TechWithTim (2019), Javilop (Javilop, 2012), ProgramArcadeGames (Craven, 2017), Pygame (Pygame, 2011a) and ZetCode (Bodnar, 2018) were extremely helpful toward the development of the project, providing alternative concepts on how to code the project in terms of the different styles of functions and loops I could use to make the code more efficient and better at running the game with fewer system resources. I found out that I should use more functions in my code to run tasks (which I had not done in the first attempt of the project), and that I should be more organised in coding the project so it would be easier to understand, something I implemented in the second attempt of the game. The courses on YouTube were also very valuable, due to the visual and auditory style of learning involved when watching these videos: Investary's Basic Python playlist (Investary, 2014), Programming with Mosh's tutorial (Hamedani, 2019), and FreeCodeCamp.org's Full Course for Beginners (Dane, 2018) helped me improve my coding skills, both when starting the first and second attempt of my project. I knew I had to use Pygame from the start of the project after eliminating TKinter as an option (as I would need to use Pygame anyways), so using the resources provided by OpenSource (Kenlon, 2017) and InventWithPython (Sweigart, 2019) helped in this part of the development of my programming knowledge, as these guides and tutorials focused solely on the modules that I was using, allowing me to directly transfer the exact skills I developed from them into my code.

I also found out every detail about the Tetris game using the official Tetris wiki (Tetris Wiki, 2001), including the exact colours used, shape and board size, falling speed, point system, etc. (Fig. A) rather than just relying on the parts that I experienced in the game, in order to get a rounded sense for the workings of the insides of the game. I created a document outlining what I intended to implement into the game, and what to exclude based on the initial perceived difficulty of creating the feature (Fig. A).

Since most of my secondary research consisted of learning a language rather than researching hard content, I have mentioned the resources that I used, but I cannot

pinpoint exactly every little bit of syntax or language that I learned going through the online resources, as this was a continuous process throughout the project. I can only mention the sites where I got the knowledge from. Knowing how to code VB.NET prior to starting this project helped me greatly, as I knew a little bit about coding from before; I converted those coding skills using a guide by Raspberry Pi Foundation which teaches VB.NET programmers to code in Python (Raspberry Pi Foundation, 2017).

Target Audience

My target audience is people ages 6+ who have a Windows computer. This is because the aim of the project was to create an executable (.exe) file which anyone with a Windows device can use, to try out the game and to try and beat the score of the artificial intelligence, at least initially. Since Tetris has been sold over 170 million times (Wikipedia Contributors, 2019a), there is a very wide audience that can recognize the game; the original Tetris game is aimed at children 6+ (LearningWorks for Kids, 2012a), so my version of the game is targeted at the same age group as well.

Project making

First Attempt

I attempted to program the game using my limited experience and knowledge about coding non-console-based programs, fixing major flaws in the framework of the game as the problems came along – this proved to be fatal to the first attempt of my game.

Before starting to code the project, I looked into useful modules that I could use when coding the game; I found the Pygame module to be fundamental, and others such as "random" to be useful to select which pieces to generate, later on in the code ((Python Wiki, 2018a), (Python Tips, 2013)). While coding the project, I often referred back to the tutorials, so I knew how to code exactly what I needed in order for the game to work. The Pygame documentation was also very useful in this regard (Pygame, 2011).

```
import pygame, random

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
WIDTH = 20
HEIGHT = 20
MARGIN = 2
grid = []
for row in range(24): ##making the grid
    grid.append([])
    for column in range(10):
        grid[row].append(0)

pygame.init()
screensize = [255, 255]
screen = pygame.display.set_mode([222, 530])
pygame.display.set_caption("Tetris")

done = False

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
            pygame.quit()

    for row in range(24): #draw colours
        for column in range(10):
            color = WHITE
            pygame.draw.rect(screen,
                             color,
                             [(MARGIN + WIDTH) * column + MARGIN,
                              (MARGIN + HEIGHT) * row + MARGIN,
                              WIDTH,
                              HEIGHT])
            pygame.display.flip()
```

Fig. 1 – code to draw the initial blank grid

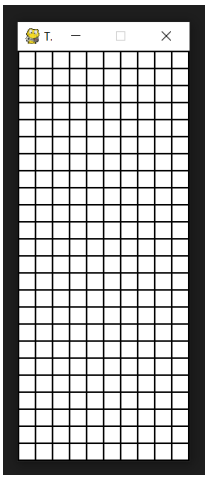

Fig. 2 – blank grid

```
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
            pygame.quit()
    if randomshape == 1 : #Oshape
        grid[ycoord - 1][xcoord] = randomshape
        grid[ycoord][xcoord] = randomshape
        grid[ycoord - 1][xcoord + 1] = randomshape
        grid[ycoord][xcoord + 1] = randomshape

    elif randomshape == 2 : #Ishape
        grid[ycoord - 3][xcoord] = randomshape
        grid[ycoord - 2][xcoord] = randomshape
        grid[ycoord - 1][xcoord] = randomshape
        grid[ycoord][xcoord] = randomshape

    elif randomshape == 3 : #Jshape
        grid[ycoord - 2][xcoord] = randomshape
        grid[ycoord - 1][xcoord] = randomshape
        grid[ycoord][xcoord] = randomshape
        grid[ycoord][xcoord - 1] = randomshape

    elif randomshape == 4 : #Lshape
        grid[ycoord - 2][xcoord] = randomshape
        grid[ycoord - 1][xcoord] = randomshape
        grid[ycoord][xcoord] = randomshape
        grid[ycoord][xcoord + 1] = randomshape
```

Fig. 3 – code to draw each individual shape in relation to the "home coordinates"

Firstly, I created a blank grid (Fig. 2), on which the pieces would fall, using a 2-dimensional array in which each element/gridblock is assigned a value of "0" (lines 9 – 12 in Fig. 1); this would be drawn on the grid using the "pygame.draw.rect()" function, which would colour in the grid with the predetermined colour "WHITE", as each element in the array has a value of "0" . The size of the margins is defined by the constant "MARGIN" and are coloured black. The code can be seen in Fig. 1, and the program that is run can be seen in Fig. 2. I was already getting stuck at some parts of this stage, as getting started was the hardest part, so I used some help from Stack Overflow for specific questions like how to draw draw rectangles (2013).

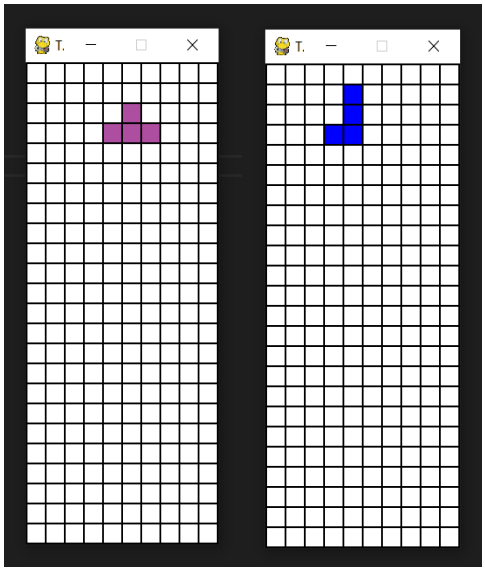

Fig. 4 – single shape drawn on the grid

Next, I decided to use fixed coordinates on the board for the "home block" at the top of the board, and then coloured additional blocks in relation to the home block to make shapes. For example, for the "I" piece, which is 4 blocks stacked on top each other, I created an "if" statement that coloured the coordinates (x, y), (x, y – 1), (x, y – 2) and (x, y – 3), shown by the lines 44 – 48 in Fig. 3. I did this for all 7 shapes and using the "random.randint" function, I assigned a random number to the variable "randomshape". I assigned each shape a number, so if "randomshape" was 1, then the program would colour in the "O" shape and if "randomshape" was 2, then the program would colour in the "I" shape, etc. for each piece. Since the "random.randint" function was outside the main "while" loop, it only generated one piece; this can be seen in Fig. 4. Additionally, I assigned each piece its colour, so if the value of the "randomshape" variable was "1" it would colour the shape yellow for the "O" piece, etc. I used the official Tetris colours for the shapes in the game (Tetris Wiki, 2001). With the code in Fig. 5, the board is "checked" for changes in each element of the array, so if the value of the array remains "0",

```
80      for row in range(24): #draw colours
81          for column in range(10):
82              color = WHITE
83              if grid[row][column] == 1:
84                  color = YELLOW
85              elif grid[row][column] == 2:
86                  color = CYAN
87              elif grid[row][column] == 3:
88                  color = BLUE
89              elif grid[row][column] == 4:
90                  color = ORANGE
91              elif grid[row][column] == 5:
92                  color = RED
93              elif grid[row][column] == 6:
94                  color = PURPLE
95              elif grid[row][column] == 7:
96                  color = GREEN
97              pygame.draw.rect(screen,
98                               color,
99                               [(MARGIN + WIDTH) * column + MARGIN,
100                               (MARGIN + HEIGHT) * row + MARGIN,
101                               WIDTH,
102                               HEIGHT])
103         pygame.display.flip()
```

Fig. 5 – if statements that colour each gridblock according to the value of the element in the array

```
34  while not done:
35      for event in pygame.event.get():
36          if event.type == pygame.QUIT:
37              done = True
38              pygame.quit()
39
40      pygame.time.delay(delay)
41      ycoord = ycoord + 1
```

Fig. 6 – code that makes the block "fall" & allows the user to quit the game

it will stay white, but if the value of the number in the array is changed, the corresponding gridblock will be coloured accordingly.

In order to make the pieces fall, I made the y coordinate increment every time the program looped through the "while" loop, as shown on line 41 in Fig. 6. However, this meant that the piece would leave a trail behind it (Fig. 7), which was very problematic; this happened because the piece did not actually move, rather, it was just coloured in again after each increment of its y coordinate. I decided to tackle this problem later. At this stage of the project, I also implemented the exit function, so if the user pressed the "X" button on the window, the game would quit; the code is shown in the first 5 lines of Fig. 6 (G, 2013).

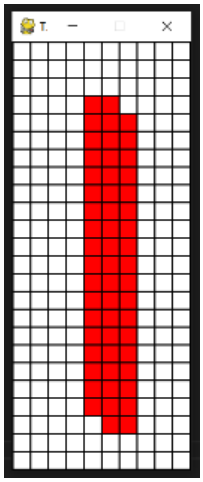

Fig. 7 – piece falling with trail behind it

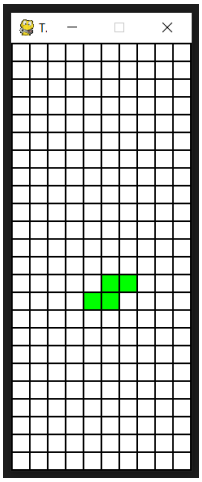

Fig. 8 – piece falling with trail coloured white

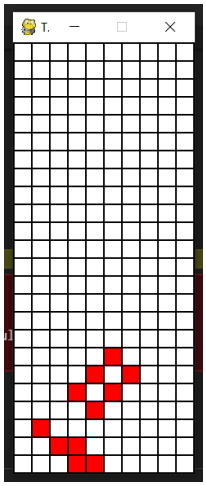

Fig. 9 – trails from piece being moved sideways

I then tried to colour the trail behind the pieces white, which worked as a temporary solution for the problem. I did this by changing the value of the elements above the piece in the array to "0", making the colour white again. The code can be seen in Fig. 10 on lines 52, 53 and 61, and the board with the piece moving is shown by Fig. 8.

```
46    if randomshape == 1 : #Oshape
47        grid[ycoord - 1][xcoord] = randomshape
48        grid[ycoord][xcoord] = randomshape
49        grid[ycoord - 1][xcoord + 1] = randomshape
50        grid[ycoord][xcoord + 1] = randomshape
51
52        grid[ycoord - 2][xcoord] = 0
53        grid[ycoord - 2][xcoord + 1] = 0
54
55    elif randomshape == 2 : #Ishape
56        grid[ycoord - 3][xcoord] = randomshape
57        grid[ycoord - 2][xcoord] = randomshape
58        grid[ycoord - 1][xcoord] = randomshape
59        grid[ycoord][xcoord] = randomshape
60
61        grid[ycoord - 4][xcoord] = 0
62
```

Fig. 10 – code that colours the trails white

```
34  while not done:
35
36      pygame.time.delay(delay)
37
38      for event in pygame.event.get():
39          if event.type == pygame.QUIT:
40              done = True
41              pygame.quit()
42          elif event.type == pygame.KEYDOWN:
43              if event.key == pygame.K_LEFT:
44                  xcoord = xcoord - 1
45              if event.key == pygame.K_RIGHT:
46                  xcoord = xcoord + 1
47
48      ycoord = ycoord + 1
```

Fig. 11 – code that converts user input to piece moving sideways piece being moved sideways

Moving the pieces is very important in the game, so I programmed the keyboard arrow keys to move the piece left and right, whose code is shown in Fig. 11 (Pygame, 2011). This again left a trail behind the pieces due to the same problem that caused the trail when the pieces fell straight down, as seen in Fig. 9.

I soon realized that the foundations I had built for the game, such as the way the pieces were generated, moved and fell, were extremely flawed. At this point, I had to make my aims for my project more realistic to achieve; these were originally to create a version of Tetris accompanied with an artificial intelligence that can play the game to a proficient level, beating the average human player. I discarded the second half of my project, which was to code an artificial intelligence that plays my game, in

order to focus solely on the production of the game itself; my new aims for the project were to create a relatively working version of the Tetris game. I then proceeded to continually improve my Python skills, again using more online resources, and then re-attempted to program the game, with far greater success and clearly visible improvements to both the inside framework of the code and the user interface of the game itself.

Second Attempt

In my second attempt, I started off the same way as the first attempt, creating a grid using an array, although I coded everything in separate functions (Fig. 12), as some functions will be needed to be called multiple times, making it easier to call a function multiple times rather than reuse the whole portion of code again; as I did not do this for the first attempt, I would have ended up repeating much of the code if I continued with it. Furthermore, in this version, boxes are drawn differently; if the board's array contains a "." (line 29 of Fig. 12) (Pygame, 2011a), the corresponding location will be coloured dark blue. This can be seen in the function: "drawBox" (line 37 of Fig. 12).

```
16  def game():
17      board = BlankBoard()
18      while True:
19          for event in pygame.event.get():
20              if event.type == pygame.QUIT:
21                  exit()
22          DISPLAY.fill((0, 10, 50))
23          drawBoard(board)
24          pygame.display.update()
25
26  def BlankBoard():
27      board = []
28      for i in range(10):
29          board.append(['.'] * 20)
30      return board
31
32  def drawBoard(board):
33      for x in range(10):
34          for y in range(20):
35              drawBox(x, y, board[x][y])
36
37  def drawBox(box_x, box_y, colour, pixel_x=None, pixel_y=None):
38      if colour == '.':
39          return
40      if pixel_x == None and pixel_y == None:
41          pixel_x = box_x * 20
42          pixel_y = box_y * 20
43      pygame.draw.rect(DISPLAY, COLOURS[colour], (pixel_x, pixel_y, 20, 20))
44
45  game()
```

Fig. 12 – the functions that draw the board & boxes

```
21  I = [['.O.',
22        '.O.',
23        '.O.',
24        '.O.',]]
25  O = [['...',
26        'OO.',
27        'OO.',
28        '...']]
```

```
13  S = [['...',
14        '.OO',
15        'OO.',
16        '...']]
17  Z = [['...',
18        'OO.',
19        '.OO',
20        '...']]
```

Fig. 13 – shape presets

I then created the pieces with "."s and "O"s, as shown in Fig. 13; this time, the pieces were coding in lists, and aligned with the elements of the array. In the function "NewPiece()", a piece is selected using "random.choice", which is defined in the beginning of the code, and if the shape matches (in the "if" statement), the colour of the shape is assigned to "tempcolour". These colours and the short names for the shapes are defined in the beginning of the code, with the colours being in RGB. Afterwards, the properties of the shape are replaced in the "dictionary" "newPiece", which is returned when the function is called. The code can be seen in Fig. 14. Since a new piece is called within the main loop, the pieces are

```
66  def NewPiece():
67      shape = random.choice(list(PIECELIST))
68      if shape == 'S':
69          tempcolour = 4
70      elif shape == 'O':
71          tempcolour = 3
72      elif shape == 'I':
73          tempcolour = 0
74      elif shape == 'T':
75          tempcolour = 5
76      elif shape == 'Z':
77          tempcolour = 6
78      elif shape == 'J':
79          tempcolour = 1
80      elif shape == 'L':
81          tempcolour = 2
82      newPiece = {'shape': shape,
83                  'x': 4,
84                  'y': 0,
85                  'colour': tempcolour}
86      return newPiece
```

Fig. 14 – code that assigns properties for a new piece

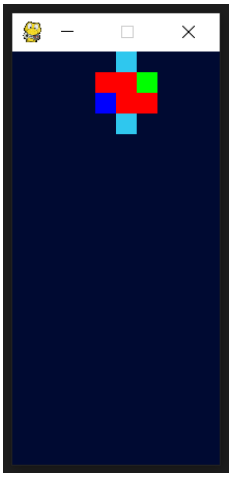

Fig. 15 – pieces being generated continuously

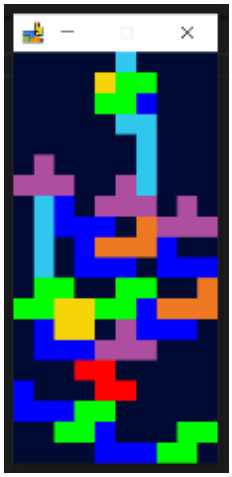

Fig. 16 – the final state of the game, with the pieces moving properly

continually generated despite there being no space for another piece to be made, shown by Fig. 15. At this stage, the pieces did not move down or sideways.

There is one major difference between the game I coded previously and this version: I coded the game so that the pieces are generated above the board instead of directly moving on the board; this solution solves the problem of the previous attempt: the moving of the pieces left and right. Now, when you press the arrow keys, the shape moves in the direction of the key press; this is the same when the piece moves down, eliminating any trails left behind by the shape. This is because the shape is one unit that is predetermined in the beginning of the code meaning the shape itself can really go down, whereas in the previous version, the shape would be redrawn every time the program looped.

To draw the shape and its colour onto the board, if the array's value changes to an "O", such as when the pieces land, the board will change colour according to a predetermined colour using the property of the shape "colour".

I finally reached the to-be end stage of my game (Fig. 16), where the pieces stack how they are meant to, are generated randomly, while using the correct colours that the original game uses as well as many other small features such as the logo at the top right of the window (Icons, 2019). Some of the major components that I could not implement into the game were the rotation systems, as well as the main focus of the game, which is clearing the lines and the scoring system; this was partly due to time restrictions, but also due to the restrictions of my coding skills, despite the vast amount I learned through the journey of the project. Having focused on solely the game, and learning the language and the nuances of the Pygame module from the start would have accelerated my growth and allowed me to produce a version of the game with far more of the features I had planned; this was the major flaw in my project stopping it from growing to its full potential.

Evaluation

Skills and Project Development

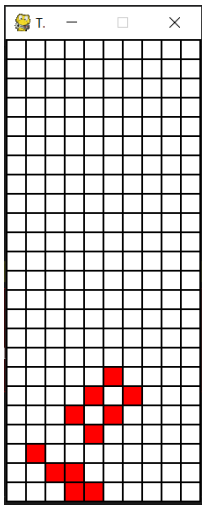

Fig. 17 – the first attempt of the game

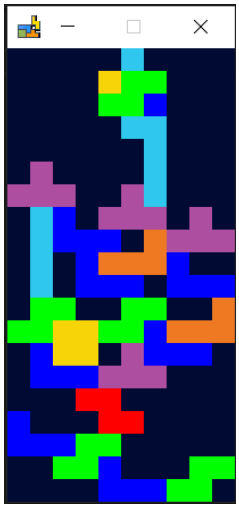

Fig. 18 – the second attempt of the

One major skill that I improved on during the EPQ was my programming skills; I started off this project knowing next to nothing about coding in Python. Using online resources, I have independently learned to code, and applied that code in order to produce the final artefact: a semi-working version of Tetris. This is the core foundation of my project, and with a bit more practice, these skills could take this project and future projects even further in terms of quality, functionality and design. I learned how to solve runtime errors faster, as well as how to troubleshoot both common and uncommon errors such as syntax mistakes that a beginner programmer would make in Python; for example, it took me a couple hours to figure out that I was missing a semicolon after the first "if statement" I created in Python. Learning these skills made the game more stable and efficient. I found out that I learned the most not from the coding tutorials, online resources, and code analysis, rather, from the coding itself, and making mistakes and figuring out how to solve them. Despite this, without the coding tutorials and online resources, I would have never gotten to the level to where I could learn by just programming. My program improved from looking like Fig. 16, causing errors and leaving trails when the pieces were falling, to looking like the second image, where the pieces not only stack but the game also looks more aesthetically pleasing while being more functional at the same time.

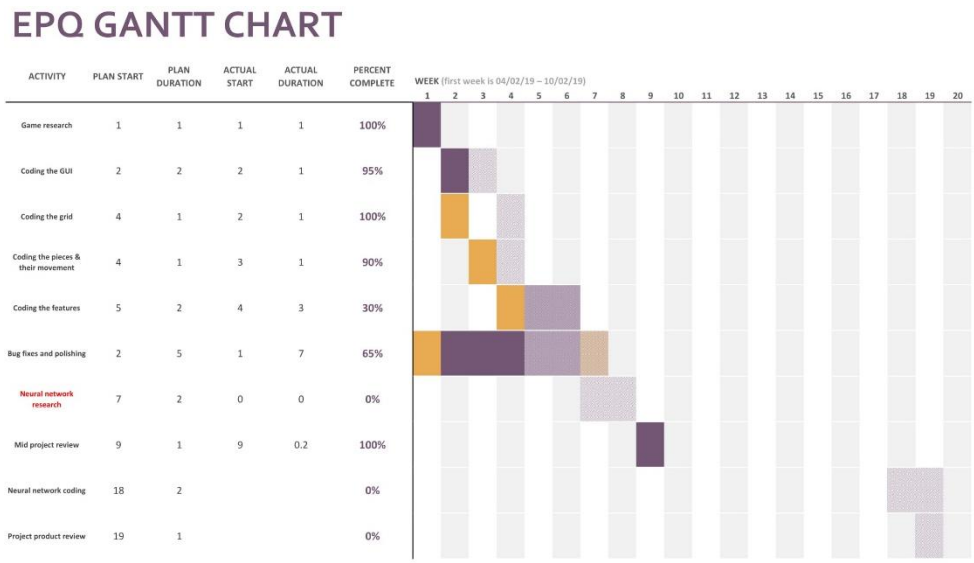

Fig. 19 – my initial Gantt chart

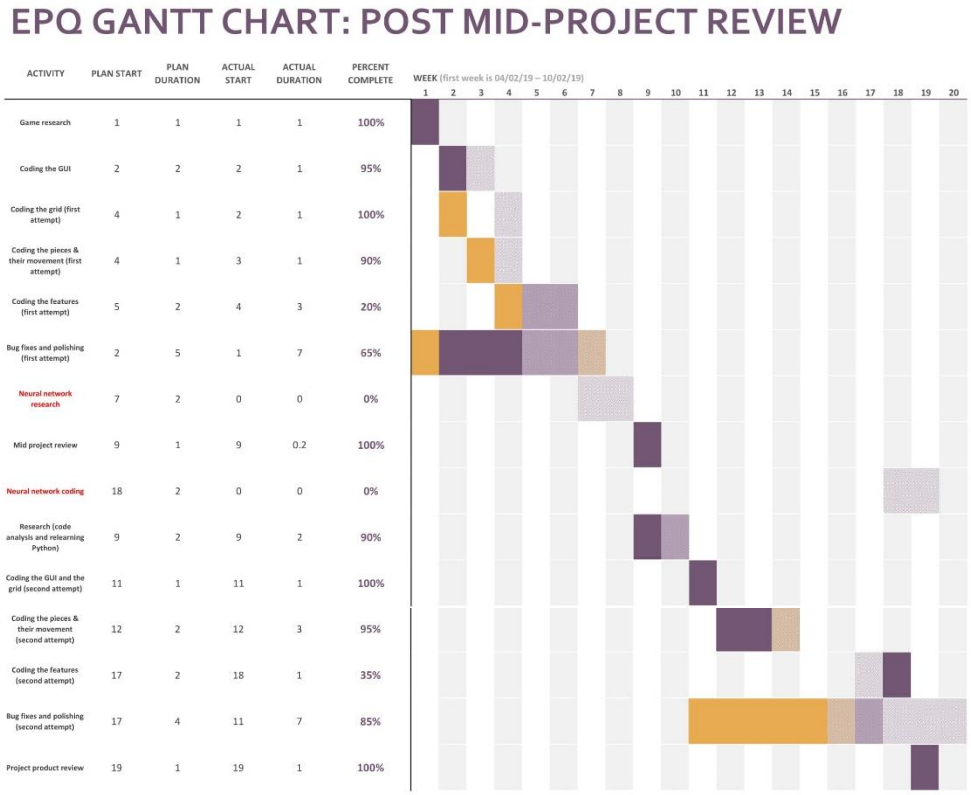

Fig. 20 – my revised Gantt chart

I have also learned to manage my time better from this EPQ project. I had initially planned the project by squeezing everything into a short time frame without any room for error, expecting everything to go well in the project (as shown in Fig. 19). I had not accounted any time for the mistakes I was eventually bound to make. Of course, I made mistakes and fell behind my schedule, so I had to heavily modify it before continuing the project after the mid project review – I allocated far more time for the coding of the game in the Gantt chart in Fig. 20. Now, I know to not under allocate time for any part

of the project, and to allow some time for mistakes and the unexpected barriers that are bound to come up in the progression of any venture.

Furthermore, I learned to do research efficiently and astutely, allowing me to spend more time learning Python and coding my program and gathering enough data about how the game works to get an idea of how to program it.

Challenges & Successes

One of the major challenges for the production of this game was the learning of Python, but more importantly, implementing what I learned into my code. This was a relatively new topic for me, and I started out as a beginner in this type of programming, which is very dissimilar to programming simple console-based programs using Visual Basic, which is what I had experience in. This meant that after doing the first round of research into the language of Python, I did not have enough experience whereas I thought I did. My learning improved greatly before I started the second attempt of the project, as I now recognized the importance of learning the language before attempting to almost blindly code in it; since I was more focused this time, my learning and progression in the online courses happened a lot faster than the first time, causing a better outcome for the project. Not using my initial resources to their full potential proved to be a challenge at the start of the project which I could not overcome.

During the first attempt of the game, the largest challenge was to get the pieces to move sideways and downwards, but because of the way that I coded the project, this was not possible. In the second attempt, the largest challenge was trying to get the pieces to rotate, and the lines to clear when the blocks were full; I also tried to implement a scoring system, but without the features mentioned prior, this would be impossible. To improve upon this, I would need to further improve my Python skills; I would achieve this by practicing coding more often and trying out new coding techniques and modules that I am not familiar with to improve my code in general. I would also require more time, something that wasn't put to its full use in the start of the project.

Some of the successes of the project were the speed at which I changed and improved my game, as constantly working on it to make it even slightly better. I also succeeded in creating the pieces and making them move properly at an even time interval, as well as displaying the board. In terms of outside the making of the actual project, my success was far greater; my project plan was detailed which allowed me to know exactly what to do at each stage of making the project, so I knew when I was falling behind. This helped me gauge the progress of my project at each stage and evaluate the advancement of the game, so I knew when to work harder on the project to get it done quicker. I also had specific goals in mind, which further helped with the point above, as I did not have to waste time on major decisions about the project. Giving myself a plethora of initial resources meant that I knew exactly what to do from the start so I would not waste time looking for resources to help me get started. This was very helpful, as the initial research took a very long time to do, so doing research simultaneously with the project would have been a disaster.

One of the benefits of a project of this nature is the speed at which one is able to alter the code, and access a plethora of resources in order to assist oneself while coding the project; for example, a DT project involving the making of a product would take far more time and effort in order to change a minute detail of the project, such as the colour of wood used to make a table, whereas the type of this project allowed me to amend and adapt my project almost instantaneously, helping me to progress very quickly through some aspects of the coding.

Conclusion

Comparing my projects to the other professional level artefacts, there are some major differences in the game, such as score, clearing of lines, etc. but with more experience, the problems and the lack of such features could be ironed out. As mentioned above, I had to omit the second part of my project, due to a misjudgement of the difficulty of embarking on such a project, but that did not in any way hinder my learning experience from undertaking this project. I did not meet my initial goal, of creating a version of Tetris as well as an AI that plays the game, but I did meet my objective after modifying it after the mid project review, of making a relatively working version of the game and improving my proficiency to code in Python and programming in general.

In conclusion, I would advise others undertaking a similar project to not make the same mistakes that I made when I undertook the project, such as overambition and poor planning. Choosing the difficulty of the project wisely is very important, as is giving yourself plenty of time in the beginning of the project to get familiar with the language you are coding in the ensure that you code the project in the best way you can. I would advise others to make sure initially that they know enough code to start the programming of the project, rather than only learning it alongside the code. This is where I blundered in this project, and it is a mistake that others should certainly be avoiding. I would also advise others to be sure that the project they are choosing is one that they are truly passionate about, since the duration of the project is quite long, and getting demotivated midway through the project would be disastrous towards the project and the work they have put in until then. Additionally, I would advise others to use all of their initial resources to their full advantage. Working on the project more continuously is important, rather than working on the project for just a few days and then leaving it until a later date; this discontinuity can ruin the flow of any project and is definitely what happened to me. Working on the project continuously would allow anyone to achieve more in the same time frame.

References

Agrawal, A. (2019a). *EPQ Timeline*. Unpublished work.

Agrawal, A. (2019b). *EPQ Timeline - Post Mid Project Review*. Unpublished work.

AI learns to play Google Chrome Dinosaur Game || Can you beat it?? (2018). *YouTube*. Available at: https://www.youtube.com/watch?v=sB_IGstiWlc [Accessed 12 Oct. 2019].

Bodnar, J. (2018). *Java Tetris*. [online] Zetcode. Available at: http://zetcode.com/tutorials/javagamestutorial/tetris/ [Accessed 11 Jun. 2019].

Christopher Castiglione (2019). *An Introduction for Beginners: Python vs Java - Learn to code in 30 Days*. [online] Learn to code in 30 Days. Available at: https://learn.onemonth.com/python-vs-java/ [Accessed 8 Oct. 2019].

Clay Mathematics Institute of Cambridge, Massachusetts (2013). *The Millennium Prize Problems | Clay Mathematics Institute*. [online] Claymath.org. Available at: https://www.claymath.org/millennium-problems/millennium-prize-problems [Accessed 13 Oct. 2019].

Craven, P.V. (2017). *Program Arcade Games with Python and Pygame*. [online] Program Arcade Games. Available at: http://programarcadegames.com/index.php?lang=en&chapter=array_backed_grids [Accessed 25 Apr. 2019].

Dane, M. (2018). *Learn Python - Full Course for Beginners [Tutorial]*. *YouTube*. Available at: https://www.youtube.com/watch?v=rfscVSovtbw [Accessed 11 Feb. 2019].

Deven Joshi (2018). *The beginner's dilemma: Should I learn Java or Python?* [online] Medium. Available at: https://medium.com/@dev.n/the-beginners-dilemma-should-i-learn-java-or-python-7efed89dc5b1 [Accessed 8 Oct. 2019].

Edx (2018). *DEV236x*. [online] Edx. Available at: https://courses.edx.org/courses/course-v1:Microsoft+DEV236x+3T2018/course/ [Accessed 19 Feb. 2019].

ELECTRONIC ARTS (2009). *TETRIS*. [online] Google.com. Available at: https://play.google.com/store/apps/details?id=com.ea.game.tetris2011_na&hl=en.

Fwend (2017). *Tetris/Java - Rosetta Code*. [online] Rosetta Code. Available at: https://rosettacode.org/wiki/Tetris/Java [Accessed 5 Apr. 2019].

G, D. (2013). *How to get keyboard input in pygame?* [online] Stack Overflow. Available at: https://stackoverflow.com/questions/16044229/how-to-get-keyboard-input-in-pygame [Accessed 27 Aug. 2019].

Gibney, E. (2017). Self-taught AI is best yet at strategy game Go. *Nature*. [online] Available at: https://www.nature.com/news/self-taught-ai-is-best-yet-at-strategy-game-go-1.22858 [Accessed 3 Sep. 2019].

Hamedani, M. (2019). *Python Tutorial for Beginners [Full Course] 2019*. [online] YouTube. Available at: https://www.youtube.com/watch?v=_uQrJ0TkZlc [Accessed 31 Aug. 2019].

Icons, T. (2019). *Tetris free icon*. [online] Flaticon. Available at: https://www.flaticon.com/free-icon/tetris_528110 [Accessed 27 Aug. 2019].

Investary (2014). *Basic Python*. *YouTube*. Available at: https://www.youtube.com/playlist?list=PLB0701884E5AE1B45 [Accessed 19 Feb. 2019].

Jared (2013). *Pygame Drawing a Rectangle*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle [Accessed 27 Aug. 2019].

Javilop (2012). *Tetris tutorial in C++ platform independent focused in game logic for beginners | Javilop*. [online] Javilop. Available at: http://javilop.com/gamedev/tetris-tutorial-in-c-platform-independent-focused-in-game-logic-for-beginners/ [Accessed 11 Jun. 2019].

Kenlon, S. (2017). *Build a game framework with Python using the Pygame module*. [online] Opensource. Available at: https://opensource.com/article/17/12/game-framework-python [Accessed 27 Aug. 2019].

LearningWorks for Kids. (2012). *Tetris - LearningWorks for Kids*. [online] Available at: https://learningworksforkids.com/playbooks/tetris/ [Accessed 9 Sep. 2019].

LearnPython (n.d.). *Learn Python - Free Interactive Python Tutorial*. [online] LearnPython. Available at: https://www.learnpython.org/ [Accessed 19 Feb. 2019].

MakeUseOf (2017). *10 Most Common Programming and Coding Mistakes*. [online] MakeUseOf. Available at: https://www.makeuseof.com/tag/common-programming-coding-mistakes/ [Accessed 12 Oct. 2019].

New, D. (2019). *Tetris gets a curious new logo for its birthday*. [online] Thumbsticks. Available at: https://www.thumbsticks.com/tetris-anniversary-new-logo-06062019/ [Accessed 27 Aug. 2019].

Pygame (2011). *Pygame Front Page*. [online] Pygame. Available at: https://www.pygame.org/docs/ [Accessed 27 Aug. 2019].

Pygame (2019a). *all*. [online] Pygame. Available at: https://www.pygame.org/tags/all [Accessed 27 Aug. 2019].

Pygame (2019b). *Tetris*. [online] Pygame. Available at: https://www.pygame.org/tags/tetris [Accessed 19 Feb. 2019].

Python Tips (2013). *20 Python libraries you can't live without*. [online] Python Tips. Available at: https://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without/ [Accessed 27 Aug. 2019].

Python Wiki (2018). *UsefulModules*. [online] Python Wiki. Available at: https://wiki.python.org/moin/UsefulModules [Accessed 27 Aug. 2019].

Raspberry Pi Foundation (2017). *Python for VB Programmers*. [online] GitHub. Available at: https://github.com/raspberrypilearning/python-for-vb-programmers/blob/master/worksheet.md [Accessed 19 Feb. 2019].

Real Python (2018). *8 World-Class Software Companies That Use Python*. [online] Real Python. Available at: https://realpython.com/world-class-companies-using-python/ [Accessed 9 Sep. 2019].

SoloLearn (2019). *SoloLearn: Learn to Code*. [online] SoloLearn. Available at: https://sololearn.com/ [Accessed 19 Feb. 2019].

Sweigart, A. (2019). *Pygame Basics*. [online] Invent with Python. Available at: http://inventwithpython.com/pygame/chapter2.html [Accessed 23 Jun. 2019].

Tetris Wiki (2000a). *Ghost piece*. [online] Tetris Wiki. Available at: https://tetris.fandom.com/wiki/Ghost_piece [Accessed 19 Feb. 2019].

Tetris Wiki (2000b). *Next*. [online] Tetris Wiki. Available at: https://tetris.fandom.com/wiki/Next [Accessed 19 Feb. 2019].

Tetris Wiki (2000c). *Random Generator*. [online] Tetris Wiki. Available at: https://tetris.fandom.com/wiki/Random_Generator [Accessed 19 Feb. 2019].

Tetris Wiki (2000d). *SRS*. [online] Tetris Wiki. Available at: https://tetris.fandom.com/wiki/SRS [Accessed 19 Feb. 2019].

Tetris Wiki (2001). *Tetris Guideline*. [online] Tetris Wiki. Available at: https://tetris.fandom.com/wiki/Tetris_Guideline [Accessed 19 Feb. 2019].

Tim (2019). *Python Tutorial*. [online] Tech With Tim. Available at: https://techwithtim.net/tutorials/game-development-with-python/tetris-pygame/tutorial-1/ [Accessed 11 Jun. 2019].

Wikipedia Contributors (2019a). *Human–computer chess matches*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Human%E2%80%93computer_chess_matches [Accessed 12 Oct. 2019].

Wikipedia Contributors (2019b). *Tetris*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Tetris [Accessed 12 Oct. 2019].