**2D orbiting relative to Earth.py**

* **Language**: Python
* **Purpose**: Simulates orbits of celestial bodies in our solar system relative to Earth.
* **Scale**: 331 lines, 8 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses functions to set graphical elements, such as set\_center and set\_data, and to define celestial bodies. Calculates new position in each animation frame.

**2D orbiting.py**

* **Language**: Python
* **Purpose**: Simulates planetary orbits within our solar system.
* **Scale**: 263 lines, 7 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Includes functions to calculate planet positions and an array to track orbital paths.

**A-star.py**

* **Language**: Python
* **Purpose**: Implements A\* search algorithm on a graph structure.
* **Scale**: 85 lines, 4 functions, 1 class
* **Object-Oriented Concepts**: Yes, uses a class to define graph nodes.
* **Data Structures/Algorithms**: Utilizes a dictionary for links between nodes, a set for input validation, and a list to manage the pathfinding process.

**Black Jack Game.py**

* **Language**: Python
* **Purpose**: Simulates a blackjack game with betting, card dealing, and scoring.
* **Scale**: 346 lines, 45 functions, 7 classes
* **Object-Oriented Concepts**: Yes, extensive use of classes to represent game components like Player, Dealer,Game, and Card. Inheritance is used for action and attributes that apply to both the dealer and the player (they both inherit Person). Encapsulation added to prevent the user from cheating using the terminal. Composition is also used with Game containing Player and Dealer which both contain Card.
* **Data Structures/Algorithms**: Manages hands with lists, organizes a deck structure, and includes methods for handling card resets and game states.

**Bouncing ball sim.py**

* **Language**: Python
* **Purpose**: Simulates a bouncing ball within a confined environment.
* **Scale**: 70 lines, 2 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses basic plotting functions to adjust simulation visuals, setting graph limits and titles for clarity.

**Breadth first.py**

* **Language**: Python
* **Purpose**: Implements the Breadth-First Search algorithm.
* **Scale**: 28 lines, 1 function, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses Breadth-First Search for node traversal.

**Colliding ball sim.py**

* **Language**: Python
* **Purpose**: Simulates the collision of balls within a confined space.
* **Scale**: 180 lines, 9 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Implements basic physics functions to manage ball velocity, collision detection, and response.

**Demo\_game.py**

* **Language**: Python
* **Purpose**: Prototype for a simple game with player controlling a ship that moves around in our solar system.
* **Scale**: 522 lines, 10 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses functions to manage player actions and game state. Builds on many of the aspects of the previous orbiting simulations while implementing them in Pygame.

**Depth-first.py**

* **Language**: Python
* **Purpose**: Implements the Depth-First Search algorithm.
* **Scale**: 30 lines, 1 function, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Utilizes Depth-First Search for traversal in a graph structure.

**Dijkstra's.py**

* **Language**: Python
* **Purpose**: Implements Dijkstra's algorithm for shortest path calculation.
* **Scale**: 29 lines, 1 function, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses Dijksta’s algorithm. Uses sorting algorithms and a dictionary to manage node distances.

**Earth Collision Simulation.py**

* **Language**: Python
* **Purpose**: Simulates collision paths of an Earth and another Earth clone travelling the opposite direction in an identical orbit.
* **Scale**: 179 lines, 12 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Combines many aspects seen in other simulations.

**Falling Sand prototype.py**

* **Language**: Python
* **Purpose**: Prototype simulation of different pixelated particles in different states in a constrained environment where the user can place particles and change temperatures (this is my Computer Science A-Level coursework and will soon be finished once full and friendly user interface is added)
* **Scale**: 807 lines, 1 function, 20 classes
* **Object-Oriented Concepts**: Yes, uses classes to represent particles. Uses inheritance to streamline the creation and addition of new particles (e.g: *Sand* inherits *Solid* which inherits *Particle*).
* **Data Structures/Algorithms**: Implements multiple classes to manage particle physics and behavior. Uses a 2D array to store the position of the particles in a grid, then iterates a secondary list containing only the particles and no empty spaces for simulation and animation purposes.

**Hole through Planet simulation.py**

* **Language**: Python
* **Purpose**: Simulates an object moving through a hypothetical hole in the Earth.
* **Scale**: 180 lines, 9 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses function to update positions in each frame and uses lists to create a trail of where the ball is going while also rendering two plot animations at once.

**Pygame Game.py**

* **Language**: Python
* **Purpose**: A simple two player game prototype using Pygame library.
* **Scale**: 145 lines, 5 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Uses Pygame's built-in structures to handle game rendering and event handling.

**RLE project.py**

* **Language**: Python
* **Purpose**: Implements Run-Length Encoding for data compression.
* **Scale**: 156 lines, 3 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Basic RLE compression logic with functions for encoding and decoding.

**Simple Waves Simulation.py**

* **Language**: Python
* **Purpose**: Simulates simple wave motion in a graphical environment.
* **Scale**: 51 lines, 0 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**sorting.py**

* **Language**: Python
* **Purpose**: Demonstrates merge sort.
* **Scale**: 44 lines, 2 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Implements merge sort using recursion.

**Usain Bolt and Kinematics.py**

* **Language**: Python
* **Purpose**: Simulates Usain Bolt’s running motion using kinematic equations.
* **Scale**: 106 lines, 1 function, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Kinematic calculations for velocity and acceleration over time.

**BPHO Challenge 1.py**

* **Language**: Python
* **Purpose**: Model for a drag-free projectile
* **Scale**: 129 lines, 4 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Used iterative Verlet method of discrete time steps to simulate trajectory.

**BPHO Challenge 2.py**

* **Language**: Python
* **Purpose**: Analytical model of projectile trajectory using x coordinates
* **Scale**: 129 lines, 4 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: All BPHO challenges build on aspects present in previous versions and simulations

**BPHO Challenge 3.py**

* **Language**: Python
* **Purpose**: Models projectiles that pass through a certain x,y coordinate entered by the user and finds the minimum launch speed alongside the low ball and high ball trajectories
* **Scale**: 248 lines, 5 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**BPHO Challenge 4.py**

* **Language**: Python
* **Purpose**: Model compares user entered trajectory with maximum range trajectory.
* **Scale**: 196 lines, 5 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**BPHO Challenge 5.py**

* **Language**: Python
* **Purpose**: Builds on challenge 3 model, adds bounding parabola, minimum launch speed trajectory, max range and high and low ball curves.
* **Scale**: 298 lines, 5 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**BPHO Challenge 6.py**

* **Language**: Python
* **Purpose**: Updated challenge 4 model by adding distance travelled by projectile
* **Scale**: 214 lines, 6 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**BPHO Challenge 7.py**

* **Language**: Python
* **Purpose**: Plotted distance of projectile from launch point against time and plotted local maximums and minimums that occurred.
* **Scale**: 327 lines, 4 functions, 1 class
* **Object-Oriented Concepts**: Yes, employs a *ball* class to organize projectiles.
* **Data Structures/Algorithms**: Used iteration and classes to streamline solution. Added functions to calculate the minimum and maximum of the graphs.

**BPHO Challenge 8.py**

* **Language**: Python
* **Purpose**: Simulated a ball bouncing
* **Scale**: 164 lines, 4 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**BPHO Challenge 9.py**

* **Language**: Python
* **Purpose**: Model with drag added alongside drag-free model
* **Scale**: 243 lines, 4 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: N/A

**BPHO Challenge Extension.py**

* **Language**: Python
* **Purpose**: Model simulating rockets with and without air resistance
* **Scale**: 389 lines, 5 functions, 0 classes
* **Object-Oriented Concepts**: No
* **Data Structures/Algorithms**: Added selection for when rockets started to burn. Breaks down advanced problem into basic problem involving Newton’s 2nd and 3rd laws by adding new variables when calculating the force to append to the force (NumPy) array. New model also adds a mass array.

**Target Shot Game**

* **Language**: Python
* **Purpose**: A game where the player aims and shoots at targets.
* **Scale**: 754 lines, 23 functions, 1 class
* **Object-Oriented Concepts**: Yes, there is a simple button class
* **Data Structures/Algorithms**: Used pickle files to hold high score board with a secondary python file (‘blank scores’). Used trigonometric functions to simulate the pulsing of the laser.

**Black Jack HTML Game**

* **Languages**: JavaScript, HTML, CSS
* **Purpose**: A browser-based blackjack game with interactive UI.
* **Scale**: Multiple files: Backend.js (469 lines, 10 functions, 7 classes) for game logic (core functions and classes), HTML for structure, and CSS for styling.
* **Object-Oriented Concepts**: Classes used in same manner as Black Jack Game.py
* **Data Structures/Algorithms**: Uses JavaScript functions to handle player actions, card dealing, and scoring. Used backend to change buttons functions and collect input boxes which allowed me to run the game using functions without having to worry about data loss. Had to create secondary functions to call object functions since it was not possible for a button to call an object function.

**Eiffel Tower**

* **Languages**: HTML, CSS
* **Purpose**: A visual webpage presenting information on the Eiffel Tower.
* **Scale**: Simple structure with HTML for layout and CSS for styling.
* **Object-Oriented Concepts**: None
* **Data Structures/Algorithms**: Primarily CSS-based for image positioning and page layout.

**Additional Links:**

* BPHO submission video: <https://youtu.be/S65eQuyCdeA>
* Advert engagement prediction: <https://www.kaggle.com/code/yasseradhair/engagement-prediction>
* Laptop pricing prediction: <https://www.kaggle.com/code/yasseradhair/laptop-pricing-predition>
* Advanced laptop pricing prediction: <https://www.kaggle.com/code/yasseradhair/advanced-laptop-pricing-prediction>