PPT 6

PPT 2

PPT 3

PPT 4

PPT 7

PPT 8

PPT 9

PPT 10

PPT 11

PPT 12

PPT 1

PPT 5

PPT 6

***Titles: Launch Physics & Trajectory in PyGame***

***Slide 1: Title Slide – Launch Physics in OMOGAME***

*“Designing Bird Launch Mechanics with Python & Math”  
Introduce the game mechanic and its real-world physics connection.*

***Slide 2: What is Physics-Based Launching?***

* *Inspired by* ***Angry Birds***
* *Launching birds using drag, angle, velocity*
* *Outcome = calculated movement under gravity  
  📌 Connects math, physics, and gameplay*

***Slide 3: Key Physics Concepts***

* ***Initial Velocity***
* ***Angle of Release***
* ***Gravity (acceleration down)***
* ***Trajectory Prediction*** *📐 Use of math.sin, math.cos, and time steps*

***Slide 4: The Bird Class – Add simulate() Method***

* *Add a method in classes.py like:*

*# Update position using velocity and gravity*

* *Simulates arc based on input drag and time*

***Slide 5: Handling Mouse Drag-and-Release***

* *Implement in main.py:*
  + *On mouse down: record start position*
  + *On mouse up: calculate velocity & angle*
  + *Launch the bird  
    💡 Use vector math to determine direction & speed*

***Slide 6: Drawing the Trajectory Arc***

* *Add function in tools.py*
* *Visual: dotted line showing future path*
* *Use frame prediction to plot arc points  
  🧪 Helps players aim before releasing*

***Slide 7: Try It Yourself – Code & Test***

*🧰 Task:*

* *Update simulate() in Bird*
* *Enable drag-and-launch in main.py*
* *Visualize path in tools.py  
  ✅ Outcome: One bird flies with correct physics + arc shown*

***Slide 8: Experiment: Change Gravity!***

* *Let students try:*

*self.gravity = 0.5 vs. .2 vs. .0*

* *Observe how arc flattens or steepens  
  🎓 Add critical thinking: “Why does the bird fall faster?”*

***Slide 9: Why Physics Makes Games Fun***

* *Predictable rules → rewarding gameplay*
* *Adds realism and skill-based challenge  
  📚 Link to real-world Newtonian motion*

***Slide 10: Recap & Project Extension***

* *✅ You implemented: drag logic, simulate(), and arc drawing*
* *Deliverables: working launcher + updated files  
  🎯 Challenge: Add wind resistance OR bounce on ground!*

PPT 6

**Slide 1: Launch Physics in OMOGAME**

**Designing Bird Launch Mechanics with Python & Math**

* OMOGAME: Siege of Avaria! introduces a physics-based bird launching mechanic that forms the core of its gameplay experience.
* The game is inspired by real-world projectile motion, allowing players to launch birds at fortress structures using intuitive drag-and-release controls.
* This mechanic directly connects mathematical concepts and physics principles to interactive gameplay, making learning engaging and hands-on.
* The implementation leverages Python and the PyGame-CE library to simulate realistic motion and collisions.
* By integrating physics into game design, OMOGAME offers a skill-based challenge that is both educational and entertaining.

**Slide 2: What is Physics-Based Launching?**

* Physics-based launching in OMOGAME draws inspiration from the popular Angry Birds game, where players control the trajectory of projectiles.
* Players initiate a launch by dragging the bird, which sets both the angle and the velocity of the shot based on the drag vector.
* Once released, the bird’s movement is calculated in real-time, simulating the effects of gravity and initial force.
* The outcome of each launch is determined by the calculated movement under gravity, resulting in a realistic parabolic path.
* This mechanic bridges the gap between mathematical modeling, physics simulation, and interactive gameplay, making it ideal for educational purposes.

**Slide 3: Key Physics Concepts**

* The initial velocity of the bird is determined by the distance and direction of the drag, which translates user input into game physics.
* The angle of release is calculated using vector mathematics, ensuring that each launch accurately reflects the player’s intended direction.
* Gravity acts as a constant downward acceleration, influencing the arc and eventual landing point of the bird.
* Trajectory prediction is achieved using trigonometric functions and discrete time steps, allowing the game to simulate motion frame by frame.
* Functions such as math.sin, math.cos, and time-based calculations are essential for updating the bird’s position throughout its flight.

**Slide 4: The Bird Class – Add simulate() Method**

* The Bird class, defined in classes.py, encapsulates the properties and behaviors of each bird in the game.
* To simulate realistic projectile motion, a simulate(self, t) method is added to the Bird class, updating its position based on velocity and gravity.
* This method calculates the bird’s new coordinates at each time step, applying the fundamental equations of motion.
* The implementation ensures that the bird follows a smooth arc, reflecting both the initial launch parameters and the ongoing influence of gravity.
* By modeling the bird’s flight path mathematically, the game achieves a high degree of realism and predictability in its launch mechanics.

**Slide 5: Handling Mouse Drag-and-Release**

* The drag-and-release mechanic is implemented in main.py, where user input is captured to determine the launch parameters.
* When the player presses the mouse button down, the game records the starting position of the drag, establishing the origin point for the launch.
* Upon releasing the mouse button, the game calculates the drag vector, which is then used to set the bird’s initial velocity and angle of release.
* The bird is launched by passing these calculated parameters to its physics simulation, initiating its flight across the game screen.
* Vector mathematics is used to translate the player’s drag gesture into precise direction and speed values for the launch.

**Slide 6: Drawing the Trajectory Arc**

* The predicted trajectory arc is visualized in tools.py, providing players with a visual guide for aiming their shots.
* A dedicated function plots a dotted line or a series of points along the predicted path, using the same physics calculations as the actual launch.
* This visualization is generated by predicting the bird’s position at multiple future time steps, effectively simulating its flight in advance.
* By displaying the trajectory arc, the game helps players understand the relationship between their input and the resulting motion.
* This feature enhances both gameplay feedback and player skill development, making the experience more interactive and educational.

**Slide 7: Try It Yourself – Code & Test**

* Students are encouraged to implement and test the complete launch physics pipeline, reinforcing their understanding through hands-on practice.
* The first step is to update the simulate() method in the Bird class to ensure accurate motion under gravity and initial velocity.
* Next, students should enable drag-and-launch input handling in main.py, allowing for interactive control of the bird’s flight.
* Visualizing the predicted arc in tools.py provides immediate feedback and helps debug the implementation.
* The goal is to achieve a scenario where a bird flies with correct physics, and its trajectory is accurately displayed before launch.

**Slide 8: Experiment: Change Gravity!**

* Students can experiment by modifying the gravity constant in the Bird class, observing how different values affect the bird’s trajectory.
* For example, setting self.gravity = 0.5, .2, or .0 will produce noticeably different arcs, from flatter to steeper paths.
* This exercise demonstrates the direct impact of gravity on projectile motion, reinforcing key physics concepts in a visual way.
* Students are encouraged to think critically about why the bird falls faster or slower with different gravity settings, linking game mechanics to real-world science1.
* Such experimentation fosters a deeper understanding of both coding and physics, making the learning process engaging and inquiry-driven.

**Slide 9: Why Physics Makes Games Fun**

* Implementing predictable physics rules creates a rewarding and skill-based gameplay experience, where players can learn and improve over time.
* Realistic motion adds depth and challenge to each launch, requiring players to master timing, angle, and power for successful shots.
* The use of physics in games like OMOGAME connects directly to Newtonian motion, illustrating concepts such as velocity, acceleration, and parabolic arcs.
* By linking gameplay to real-world science, the game makes learning both fun and relevant for students.
* This approach demonstrates how mathematics and physics can be applied creatively in technology and entertainment.

**Slide 10: Recap & Project Extension**

* In this session, you have implemented mouse drag logic for launch input, enabling interactive control over the bird’s flight.
* You have developed the simulate() method to model realistic bird motion, applying physics equations in your code.
* The trajectory arc drawing feature provides players with valuable aiming feedback, enhancing both gameplay and learning.
* As a next step, you can add new bird types with unique physics properties or experiment with game features like wind or power-ups that affect trajectory.
* Extending your project to include custom levels or new mechanics will further deepen your understanding of both PyGame and physics-based game development.

PPT 4

*Here’s a structured* ***10-slide student training presentation outline*** *based on the theme:*

***🎮 Game Modes and Flow Control in PyGame***

*Focused on Quick Game vs Basic Game and mode-switching logic in* ***OMOTEC-PyGame: Siege of Avaria!***

***✅ Slide 1: Title Slide – Game Modes & Flow Logic***

*“Understanding How Games Handle Player Turns and Rules”  
Introduce students to the concept of game modes and how flow control governs the player experience.*

***✅ Slide 2: What Are Game Modes?***

* *Two types of gameplay:*
  + *⚡* ***Quick Game*** *– First to 200 points*
  + *🧱* ***Basic Game*** *– Full structure destruction with power-ups*
* *Game behavior changes based on selected mode*

*📌 Located in game\_modes.py*

***✅ Slide 3: Why Game Modes Matter***

* *Let players choose how they want to play*
* *Increases replayability and difficulty options*
* *Good practice for writing* ***modular game logic***

***✅ Slide 4: Code Files to Focus On***

| ***File Name*** | ***Purpose*** |
| --- | --- |
| *main.py* | *Game startup, mode selection, setup* |
| *game\_modes.py* | *Logic for Quick and Basic mode* |
| *game\_screens.py* | *Display screens and menus* |

*🧩 Each file has a role in handling game flow*

***✅ Slide 5: How Quick Game Works***

* *Point-based scoring: hit blocks or aliens*
* *Each bird earns points*
* ***Winner*** *= First to 200 points*
* *Simple turn switching logic after each bird*

***✅ Slide 6: How Basic Game Works***

* *Structures must be completely destroyed*
* ***Power-ups*** *enabled (e.g., double damage, trajectory aid)*
* *Turn switching + condition checking for structure state  
  📦 Requires additional logic checks in game\_modes.py*

***✅ Slide 7: Implementing Turn Switching***

***Turn Control Logic:***

*turn = 1 if turn == 2 else 2*

*🔄 Switch players after each bird completes its path  
🎯 Can be displayed on-screen with a label like: "Now Playing: Player 1"*

***✅ Slide 8: Add a Mode Selection Menu***

*In game\_screens.py → add:*

* *Buttons: Quick Game / Basic Game*
* *Store selection as global variable or mode flag*
* *Print "Mode Selected: Quick" for testing*

*🧠 Use color-coded buttons and feedback sounds for UI enhancement*

***✅ Slide 9: Student Task – Build the Mode Manager***

*🛠️ Your practical deliverables:*

* *Update game\_modes.py with logic separation*
* *Add turn counter to alternate players*
* *Modify menu screen to select mode  
  💡 Bonus: Print score & mode at top of screen during gameplay*

***✅ Slide 10: Recap & Debugging Tips***

*✅ You should now understand:*

* *Difference between Quick & Basic mode*
* *How to alternate player turns*
* *How to select and switch modes*

*💡 Tip: Use print() often to debug game state and flow.*

PPT 4

**Slide 1: Title Slide – Game Modes & Flow Logic**

*“Understanding How Games Handle Player Turns and Rules”*

* Introduce game modes in OMOTEC-PyGame: Siege of Avaria!
* Focus: How flow control shapes player experience.
* Game inspired by Angry Birds, with two-player strategic battles.
* Learn to implement Quick and Basic modes using Pygame-CE.
* Explore turn management and mode selection for dynamic gameplay.

**Slide 2: What Are Game Modes?**

* Two gameplay types in Siege of Avaria!:
  + ⚡ Quick Game: First to 200 points by hitting blocks/aliens.
  + 🧱 Basic Game: Destroy opponent’s structure with power-ups.
* Game behavior adapts based on selected mode.
* Managed in **game\_modes.py** for modular logic.
* Enhances player choice and game variety.
* Key to replayability and strategic depth.

**Slide 3: Why Game Modes Matter**

* Allow players to choose preferred play style (fast or strategic).
* Increase replayability with varied objectives and challenges.
* Teach modular programming by separating game logic.
* Support different win conditions (points vs. destruction).
* Enable power-up integration in Basic Mode for added complexity.
* Practice for real-world game design flexibility.

**Slide 4: Code Files to Focus On**

* **main.py**: Initializes game, handles mode selection, and flow.
* **game\_modes.py**: Contains logic for Quick and Basic modes.
* **game\_screens.py**: Displays menus, mode selection, and game over screens.
* **tools.py**: Supports turn rendering and game state updates.
* Each file collaborates to manage game flow and transitions.
* Critical for seamless mode switching and player interaction.

**Slide 5: How Quick Game Works**

* Objective: First player to reach 200 points wins.
* Scoring: Points earned by hitting blocks or aliens.
* Each bird launch contributes to player’s score.
* Simple turn-switching after each bird’s trajectory ends.
* Logic in **game\_modes.py** tracks scores and win conditions.
* Fast-paced, ideal for quick matches.

**Slide 6: How Basic Game Works**

* Objective: Completely destroy opponent’s block structure.
* Power-ups enabled (e.g., double damage, full trajectory view).
* Turn-switching after each bird launch or rest.
* **game\_modes.py** checks structure health via **classes.py** Block.
* Additional logic for power-up activation and effects.
* Strategic mode with focus on planning and resource use.

**Slide 7: Implementing Turn Switching**

* Turn control logic in **game\_modes.py**:

turn = 1 if turn == 2 else 2

* Switches players after bird completes its path or stops.
* Display turn status: "Now Playing: Player 1" via **tools.py**.
* Ensures fair alternation in both Quick and Basic modes.
* Updates game state to track active player.
* Critical for maintaining two-player gameplay flow.

**Slide 8: Add a Mode Selection Menu**

* In **game\_screens.py**:
  + Add buttons for Quick Game and Basic Game.
  + Store selection in global variable or mode flag.
  + Test with print: "Mode Selected: Quick".
* Use **assets.py** to load button images from **media/**.
* Enhance UI with color-coded buttons and feedback sounds.
* Ensure **main.py** transitions to selected mode seamlessly.

**Slide 9: Student Task – Build the Mode Manager**

* **Deliverables**:
  + Update **game\_modes.py** to separate Quick/Basic logic.
  + Add turn counter for player alternation in **game\_modes.py**.
  + Modify **game\_screens.py** for mode selection menu.
  + Bonus: Display score and mode at top of screen via **tools.py**.
* Save as: my\_mode\_manager.py.
* Test functionality and submit for feedback.

**Slide 10: Recap & Debugging Tips**

* **What you learned**:
  + Differences between Quick and Basic modes.
  + Implementing turn alternation in **game\_modes.py**.
  + Creating mode selection menus in **game\_screens.py**.
  + Managing game flow across multiple modules.
* **Debugging tips**:
  + Use print() to track game state (mode, turn, score).
  + Test mode transitions incrementally in **main.py**.
  + Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 3

*Here’s a* ***10-slide student training presentation title outline*** *for the topic:*

***🎮 Game Screens & Power-Up UI in OMOGAME: Siege of Avaria!***

*Focused on integrating and managing power-ups into the Basic Game Mode using UI elements*

***✅ Slide 1: Title Slide – Powering Up the UI***

*“Strategic Power-Ups and Game Interface in PyGame”  
Introduce the UI components that enhance gameplay and how students will build & test power-up systems.*

***✅ Slide 2: Power-Ups: What and Why?***

* *Adds* ***strategic advantage*** *and variety*
* *Increases engagement and learning through experimentation*
* *Examples in game:*
  + *🎯* ***Double Damage***
  + *🛰* ***Full Trajectory View***

***✅ Slide 3: Game UI Modules Involved***

| ***Module Name*** | ***Role in Power-Up Integration*** |
| --- | --- |
| *classes.py* | *Define the PowerUp class and its attributes* |
| *game\_modes.py* | *Manage when power-ups are available and applied* |
| *assets.py* | *Load visual icons or effects* |
| *media/* | *Store icons, sounds, and visual resources* |

***✅ Slide 4: Building the PowerUp Class***

*Create PowerUp class in classes.py:*

*class PowerUp:*

*def \_\_init\_\_(self, type, position):*

*self.type = type*

*self.position = position*

*self.active = False*

*📌 Attributes: type, position, duration, active state*

***✅ Slide 5: Activate Power-Ups in Basic Mode***

*In game\_modes.py:*

* *Trigger on player action or random chance*
* *Modify bird behavior during launch or collision*
* *Handle activation flag and reset logic*

*💡 Example:*

*if powerup.type == "double\_damage":*

*bird.damage \*= 2*

***✅ Slide 6: Design Power-Up Icons & UI***

*🎨 Add to media/:*

* *double\_damage.png*
* *trajectory\_hint.png*

*🧩 In assets.py:*

*self.powerup\_icons["double\_damage"] = pygame.image.load(...)*

*🎯 Display icons when a power-up is available or active*

***✅ Slide 7: Testing in Gameplay***

*Practical Demo:*

* *Player collects or activates a power-up*
* *Power-up icon appears on screen*
* *Bird shows modified behavior  
  🧪 Test one power-up at a time for clarity*

***✅ Slide 8: Strategic Gameplay Use***

* *Players must choose* ***when*** *to use power-ups*
* *Build in* ***limited-use*** *logic (e.g., 1 per turn or per game)*
* *Encourage players to plan ahead instead of random use*

***✅ Slide 9: Student Task – Design Your Power-Up***

*👨‍🔬 Group activity:*

* *Brainstorm 1 creative power-up (e.g., triple bounce, alien freeze)*
* *Design icon + implement logic + test it  
  📂 Submit updated code and screenshot of activation*

***✅ Slide 10: Recap & Power-Up Debug Tips***

*✔ What we built:*

* *PowerUp class*
* *Visual UI elements*
* *Basic Game integration*

*💬 Tips:*

* *Use console prints to check power-up states*
* *Comment every line that changes bird behavior*

PPT 3

**Slide 1: Title Slide – Powering Up the UI**

*“Strategic Power-Ups and Game Interface in PyGame”*

* Introduce UI components enhancing gameplay in OMOTEC-PyGame: Siege of Avaria!
* Focus: Building and testing power-up systems for strategic gameplay.
* Game inspired by Angry Birds, featuring two-player fortress destruction.
* Power-ups add variety and engagement in Basic Mode.
* Students will implement and test power-ups using Pygame-CE.

**Slide 2: Power-Ups: What and Why?**

* Add strategic advantages to gameplay, increasing depth.
* Enhance player engagement through experimentation and choice.
* Examples in Siege of Avaria!:
  + 🎯 Double Damage: Doubles bird impact damage.
  + 🛰 Full Trajectory View: Shows complete bird flight path.
* Encourage creative problem-solving and replayability.
* Implemented in Basic Mode for dynamic gameplay.
* Key to differentiating Basic Mode from Quick Mode.

**Slide 3: Game UI Modules Involved**

* **classes.py**: Defines PowerUp class with attributes and behaviors.
* **game\_modes.py**: Manages power-up availability and application logic.
* **assets.py**: Loads visual icons and effects for power-ups.
* **media/**: Stores power-up icons, sounds, and visual resources.
* **tools.py**: Supports rendering power-up effects on screen.
* Collaborative module interaction ensures seamless integration.
* Critical for UI feedback and gameplay responsiveness.

**Slide 4: Building the PowerUp Class**

* Create PowerUp class in **classes.py**:
* class PowerUp:
* def \_\_init\_\_(self, type, position):
* self.type = type
* self.position = position

self.active = False

* Attributes: type (e.g., "double\_damage"), position, duration, active state.
* Defines behavior for power-ups like trajectory view or damage boost.
* Interacts with Bird class for modified launch mechanics.
* Managed by **game\_modes.py** for activation logic.

**Slide 5: Activate Power-Ups in Basic Mode**

* In **game\_modes.py**:
  + Trigger power-ups on player action or random chance.
  + Modify bird behavior during launch or collision.
  + Handle activation flag and reset logic after use.
* Example:
* if powerup.type == "double\_damage":

bird.damage \*= 2

* Ensures power-ups integrate with turn-based gameplay.
* Resets after turn to maintain game balance.
* Visual feedback via **tools.py** rendering functions.

**Slide 6: Design Power-Up Icons & UI**

* Add to **media/** folder:
  + double\_damage.png
  + trajectory\_hint.png
* In **assets.py**:

self.powerup\_icons["double\_damage"] = pygame.image.load("media/double\_damage.png")

* Display icons when power-up is available or active.
* Use **tools.py** to render icons on screen during gameplay.
* Enhances UI by providing clear visual cues.
* Icons stored in **media/** for consistent asset management.

**Slide 7: Testing in Gameplay**

* Practical demo steps:
  + Player collects or activates a power-up.
  + Power-up icon appears on screen via **tools.py**.
  + Bird shows modified behavior (e.g., doubled damage).
* Test one power-up at a time to ensure clarity.
* Verify activation in Basic Mode using **game\_modes.py**.
* Check visual feedback with **assets.py** icon loading.
* Debug using console logs to track power-up states.

**Slide 8: Strategic Gameplay Use**

* Players choose when to activate power-ups for maximum impact.
* Implement limited-use logic (e.g., one per turn or game).
* Encourage strategic planning over random activation.
* Example: Use Double Damage when targeting strong blocks.
* Managed in **game\_modes.py** for rule enforcement.
* Enhances competitive depth in two-player battles.
* Balances power-ups to avoid overpowering gameplay.

**Slide 9: Student Task – Design Your Power-Up**

* Group activity:
  + Brainstorm one creative power-up (e.g., triple bounce, alien freeze).
  + Design icon in **media/** (e.g., triple\_bounce.png).
  + Implement logic in **classes.py** and **game\_modes.py**.
  + Test in Basic Mode with visual feedback via **tools.py**.
* Deliverables: Updated code and screenshot of activation.
* Save as: my\_powerup.py.
* Submit for feedback on creativity and functionality.

**Slide 10: Recap & Power-Up Debug Tips**

* **What we built**:
  + PowerUp class in **classes.py** for behavior definition.
  + Visual UI elements with icons in **assets.py** and **media/**.
  + Basic Mode integration via **game\_modes.py**.
* **Debug tips**:
  + Use console prints to track power-up states (e.g., active, type).
  + Comment lines modifying bird behavior for clarity.
  + Test incrementally to isolate issues in **tools.py** or **game\_modes.py**.
* Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 2

*Here is a* ***10-slide student training presentation outline*** *focused on:*

***🎮 Display & Window Management in PyGame-CE***

*Learn how to set up and manage your game window for OMOGAME: Siege of Avaria!*

***✅ Slide 1: Title Slide – “Setting Up Your Game World”***

*“Build the stage before the game begins!”  
Introduce the purpose of the window and display system in a game project.*

***✅ Slide 2: Why Display Management Matters***

* *Controls* ***screen resolution****, refresh rate, and layout*
* *Dictates visual clarity, responsiveness, and performance*
* *Foundation for background, UI, characters, and effects*

*🧠 Think of the game window as your* ***digital canvas***

***✅ Slide 3: Understanding the Game Loop***

*The* ***core structure*** *of every PyGame game:*

*while True:*

*handle\_events()*

*update\_game()*

*draw\_everything()*

* *Loop runs at 60 FPS*
* *Ensures real-time rendering and input handling*

***✅ Slide 4: Initializing the PyGame Window***

*In main.py:*

*pygame.init()*

*screen = pygame.display.set\_mode((1280, 720))*

*pygame.display.set\_caption("OMOGAME: Siege of Avaria!")*

*📌 You’ve now opened a blank game window!*

***✅ Slide 5: Handling Events and Closing the Game***

*for event in pygame.event.get():*

*if event.type == pygame.QUIT:*

*running = False*

*💡 Without this, your game won’t respond to clicks or key exits.*

***✅ Slide 6: Displaying a Background Image***

* *Place an image in the media/ folder (e.g., background\_sky.png)*
* *Load it using assets.py or directly in main.py:*

*bg\_img = pygame.image.load("media/background\_sky.png")*

*screen.blit(bg\_img, (0, 0))*

*🌄 Background sets the mood and theme.*

***✅ Slide 7: Controlling the Frame Rate***

*Ensure smooth gameplay using:*

*clock = pygame.time.Clock()*

*clock.tick(60)*

* *Locks the update rate to* ***60 frames per second***
* *Prevents flickering and lag*

***✅ Slide 8: Experiment – Custom Window Design***

*💡 Try this with your team:*

* *Change window size: 800x600, 1920x1080*
* *Try different backgrounds (sky, battlefield, jungle)*
* *Add background music or ambient sound (bonus!)*

***✅ Slide 9: Student Task – Build Your Own Game Start Screen***

*✅ Deliverables:*

* *Functional game loop*
* *A loaded image background*
* *Exit button working properly*
* *Customized window title*

*📂 Save as: my\_game\_start.py*

***✅ Slide 10: Recap & Design Tips***

*✅ What you learned:*

* *Setting up display window in PyGame*
* *Game loop structure*
* *Loading + rendering backgrounds*
* *Frame rate control*

*💡 Tip: Your game’s first impression begins with a clean, responsive window.*

PPT 2

**Slide 1: Title Slide – “Setting Up Your Game World”**  
*“Build the stage before the game begins!”*

* Introduce the game: OMOTEC-PyGame: Siege of Avaria!, inspired by Angry Birds.
* Purpose: Set up the window and display system for a game project.
* Two-player strategic battle launching birds to destroy fortresses.
* Built using Pygame-CE, a Python library for game development.
* Think of the game window as your digital canvas for visuals and interactions!

**Slide 2: Why Display Management Matters**

* Controls screen resolution, refresh rate, and layout for optimal visuals.
* Ensures clarity, responsiveness, and smooth performance.
* Foundation for rendering backgrounds, UI, characters, and effects.
* Poor display setup causes lag, flickering, or visual glitches.
* Integrates assets like bird images, blocks, and power-ups from assets.py.
* Sets the stage for immersive gameplay in Siege of Avaria!

**Slide 3: Understanding the Game Loop**

* Core structure of every PyGame game:

while True:

handle\_events()

update\_game()

draw\_everything()

* Runs at 60 FPS for real-time rendering and input handling.
* Manages player inputs (mouse drags, clicks) and game state updates.
* Draws game elements like birds, blocks, and aliens from classes.py.
* Orchestrated in main.py to control game flow and transitions.

**Slide 4: Initializing the PyGame Window**

* Code in main.py to set up the game window:

pygame.init()

screen = pygame.display.set\_mode((1280, 720))

pygame.display.set\_caption("OMOTEC-PyGame: Siege of Avaria!")

* Creates a 1280x720 pixel window for gameplay.
* Sets a custom title to reflect the game’s identity.
* Initializes Pygame for graphics, input, and asset loading.
* First step to displaying elements like backgrounds and sprites.

**Slide 5: Handling Events and Closing the Game**

* Event handling code in main.py:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

* Captures user inputs like clicking the window’s close button.
* Ensures graceful game exit for a polished user experience.
* Supports gameplay inputs like mouse drags for bird launches.
* Critical for responsiveness in both Quick and Basic modes.

**Slide 6: Displaying a Background Image**

* Load and display a background from media/ folder:

bg\_img = pygame.image.load("media/background\_sky.png")

screen.blit(bg\_img, (0, 0))

* Managed in assets.py for loading images (birds, blocks, power-ups).
* Sets the visual theme (e.g., sky, battlefield) for immersion.
* Drawn each frame in the game loop for consistency.
* Enhances the mood of Siege of Avaria!’s strategic battles.

**Slide 7: Controlling the Frame Rate**

* Ensures smooth gameplay with consistent updates:

clock = pygame.time.Clock()

clock.tick(60)

* Locks game loop to 60 frames per second.
* Prevents flickering, lag, or uneven gameplay.
* Supports physics calculations for bird trajectories in classes.py.
* Maintains performance across various hardware setups.

**Slide 8: Experiment – Custom Window Design**

* Team challenge: Customize the game window!
* Try window sizes: 800x600, 1920x1080 in main.py.
* Experiment with backgrounds: sky, battlefield, jungle from media/.
* Optional: Add background music or ambient sound for immersion.
* Modify main.py to test changes and observe visual/performance impacts.
* Share results to compare designs and functionality.

**Slide 9: Student Task – Build Your Own Game Start Screen**

* Deliverables:
  + Functional game loop with event handling.
  + Loaded background image from media/ folder.
  + Working exit button for proper game closure.
  + Customized window title (e.g., "My Siege of Avaria").
* Save as: my\_game\_start.py.
* Test in main.py to ensure functionality.
* Submit for feedback on design and implementation.

**Slide 10: Recap & Design Tips**

* What you learned:
  + Setting up a PyGame display window in main.py.
  + Structuring a game loop for real-time updates.
  + Loading and rendering backgrounds via assets.py.
  + Controlling frame rate for smooth gameplay.
* Design tip: A clean, responsive window creates a strong first impression.
* Next steps: Explore game\_modes.py for Quick and Basic mode logic.
* Reference: Pygame Official Documentation (<https://www.pygame.org/docs/>).

PPT 7

Based on the attached **PYGAME-LEVEL3 report** and your provided learning goal, here is a structured **10-slide title set** for a student-facing training presentation focused on:

**🏗️ Procedural Structure Generation in PyGame**

*Using Python’s random & math modules to build dynamic, destructible towers in OMOGAME*

**✅ Slide 1: Title Slide – “Generating Structures with Python”**

*“Create Randomized Fortresses in PyGame with Code!”*  
Kick off the session with the power of procedural generation for gameplay variation.

**✅ Slide 2: Why Procedural Generation?**

* Infinite layout combinations = high replayability
* Makes every round of the game feel fresh
* Saves time vs manually designing each level

📌 *Used in games like Minecraft, Angry Birds, and Terraria!*

**✅ Slide 3: Tools We’ll Use**

| **Module** | **Purpose** |
| --- | --- |
| random | Randomize block positions, types, heights |
| math | Calculate spacing, angles, gravity effects |
| tools.py | Where structure functions will be written |

**✅ Slide 4: How Blocks Become Towers**

Define block shape, type, and position:

for i in range(num\_blocks):

x = random.randint(300, 600)

y = base\_y - i \* block\_height

📦 Stack blocks vertically or arrange in shapes (L, T, Pyramid)

**✅ Slide 5: Creating the Structure Generator**

In tools.py, write:

def generate\_structure(start\_x, base\_y, block\_count):

# Place blocks randomly

return block\_list

🧱 Each block = new object of Block class with health + material

**✅ Slide 6: Support for Multiple Block Types**

In classes.py, enhance Block with:

class Block:

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

if type == "wood":

self.health = 50

elif type == "stone":

self.health = 100

🎨 Use assets.py to load different block images

**✅ Slide 7: Display Your Structure**

In main.py or game\_modes.py:

* Call generate\_structure()
* Loop through blocks & draw each one with screen.blit()  
  🎯 *Test multiple builds with each run*

**✅ Slide 8: Experiment with Designs**

Student Creativity Time:

* Shape ideas: Wall, L-shape, T-shape, Pyramid
* Add block color-coding
* Bonus: Add a “Structure Preview Mode” to your menu

**✅ Slide 9: Student Task – Build a Fortress Function**

Your Challenge:

* Define a structure generator that:
  + Randomizes block count & layout
  + Uses at least 2 types of blocks
* Screenshot your structure in-game

🗂️ Deliver: tools.py, classes.py, background render

**✅ Slide 10: Recap & Design Tips**

✔ You now understand:

* What procedural generation is
* How to use random to spawn blocks
* How to visualize dynamic towers in-game

💡 Tip: Use random.seed() during testing to lock a pattern for debugging

PPT 7

**Slide 1: Title Slide – “Generating Structures with Python”**

*“Create Randomized Fortresses in PyGame with Code!”*

* Kick off: Procedural generation for dynamic gameplay in OMOTEC-PyGame: Siege of Avaria!
* Inspired by Angry Birds, players destroy randomized fortresses.
* Learn to use Python to generate varied block structures.
* Focus: Create replayable, unique layouts with Pygame-CE.
* Enhances strategic depth in two-player battles.

**Slide 2: Why Procedural Generation?**

* Creates infinite layout combinations for high replayability.
* Keeps each game round fresh and engaging.
* Saves time compared to manual level design.
* Used in games like Minecraft, Angry Birds, and Terraria.
* Implemented in **tools.py** for Siege of Avaria!’s fortresses.
* Encourages creative and unpredictable gameplay strategies.

**Slide 3: Tools We’ll Use**

* **random**: Randomizes block positions, types, and heights.
* **math**: Calculates spacing, angles, and gravity effects for blocks.
* **tools.py**: Contains structure generation functions.
* **classes.py**: Defines Block class for structure properties.
* **assets.py**: Loads block images from **media/** for rendering.
* Collaborative modules ensure dynamic fortress creation.

**Slide 4: How Blocks Become Towers**

* Define block shape, type, and position in **tools.py**:
* for i in range(num\_blocks):
* x = random.randint(300, 600)

y = base\_y - i \* block\_height

* Stack blocks vertically or arrange in shapes (L, T, Pyramid).
* Each block is a **Block** class instance with health.
* Random placement enhances strategic challenge.
* Supports varied fortress designs for both players.

**Slide 5: Creating the Structure Generator**

* In **tools.py**:
* def generate\_structure(start\_x, base\_y, block\_count):
* block\_list = []
* for i in range(block\_count):
* block\_list.append(Block(type, x, y))

return block\_list

* Generates blocks with randomized positions and types.
* Returns list of **Block** objects for rendering.
* Called in **game\_modes.py** for each player’s fortress.
* Ensures varied layouts for Quick and Basic modes.

**Slide 6: Support for Multiple Block Types**

* Enhance **Block** class in **classes.py**:
* class Block:
* def \_\_init\_\_(self, type, x, y):
* self.type = type
* self.health = 50 if type == "wood" else 100

self.pos = (x, y)

* Types: Wood (50 health), Stone (100 health).
* Load images in **assets.py**: pygame.image.load("media/wood\_block.png").
* Visual distinction for block types enhances UI.
* Health impacts gameplay in **tools.py** collision detection.

**Slide 7: Display Your Structure**

* In **main.py** or **game\_modes.py**:
  + Call generate\_structure() to create fortress.
  + Loop through blocks: screen.blit(block.image, block.pos).
* Renders blocks using images from **assets.py**.
* Test multiple builds to ensure randomization works.
* Update each frame in game loop for dynamic visuals.
* Integrates with **tools.py** for collision and damage handling.

**Slide 8: Experiment with Designs**

* Student creativity time:
  + Try shapes: Wall, L-shape, T-shape, Pyramid.
  + Add color-coding for block types (e.g., brown for wood, gray for stone).
  + Bonus: Add “Structure Preview Mode” in **game\_screens.py**.
* Modify **tools.py** to test different layouts.
* Share unique fortress designs with team.
* Experiment with block counts and patterns for variety.

**Slide 9: Student Task – Build a Fortress Function**

* **Challenge**:
  + Create a structure generator in **tools.py**.
  + Randomize block count (5–15) and layout.
  + Use at least two block types (wood, stone).
  + Capture screenshot of in-game fortress.
* **Deliverables**: Updated **tools.py**, **classes.py**, rendered background.
* Save as: my\_fortress\_generator.py.
* Submit for feedback on creativity and functionality.

**Slide 10: Recap & Design Tips**

* **What you learned**:
  + Procedural generation for dynamic fortresses.
  + Using **random** to spawn varied block layouts.
  + Visualizing towers with **assets.py** and **tools.py**.
  + Integrating structures into game loop.
* **Debugging tip**: Use random.seed(42) to lock patterns for testing.
* Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 8

Based on the attached **OMOTEC-PyGame: Siege of Avaria! report** and your objective focus, here’s a structured **10-slide training presentation title outline** for:

**🧱 Collision Detection & Damage System in PyGame**

*Training students to build destructible environments using Rect collisions in PyGame-CE*

**✅ Slide 1: Title Slide – “Detect. Damage. Destroy.”**

*Build Realistic Collisions and Breakable Structures in OMOGAME*

Introduce students to collision systems and why they matter in gameplay.

**✅ Slide 2: What is Collision Detection?**

* Tracks when two objects overlap or impact
* Used to detect bird hits on blocks or aliens
* Critical for gameplay scoring and destruction

🧠 *Imagine birds flying through thin air with no reaction... That’s what happens without collision logic!*

**✅ Slide 3: The Tools Behind Collisions**

| **Module** | **Function** |
| --- | --- |
| tools.py | Houses collision detection methods |
| classes.py | Defines block health + hit response (take\_damage) |
| assets.py | Manages visuals like crack images or debris |

**✅ Slide 4: Writing check\_block\_collisions()**

In tools.py:

def check\_block\_collisions(bird, blocks):

for block in blocks:

if bird.rect.colliderect(block.rect):

block.take\_damage(bird.damage)

🎯 Uses PyGame’s Rect.colliderect() method to trigger contact

**✅ Slide 5: Adding take\_damage() to Block Class**

In classes.py:

class Block:

def take\_damage(self, amount):

self.health -= amount

if self.health <= 0:

self.destroy()

📉 Block health decreases on hit → destroyed when health ≤ 0

**✅ Slide 6: Visual Feedback: Show Cracks or Debris**

* In assets.py, load damaged block variants
* In tools.py, draw image based on block’s health:
* if block.health < 50:
* screen.blit(cracked\_image, block.rect)

📌 Visual cues make the damage system **visible and intuitive**

**✅ Slide 7: Alien Collision Detection**

Write check\_alien\_collisions():

* Similar to block collisions
* On hit, alien disappears and bonus score is awarded
* Trigger animation or sound effect for feedback

**✅ Slide 8: Test and Debug Your System**

🧪 Test Case:

* Launch a bird → hits a stone block
* Check if:
  + Health reduces
  + New sprite appears
  + Block eventually disappears

💬 Use print(block.health) in terminal to verify logic

**✅ Slide 9: Student Task – Build a Damage-Test Arena**

🛠️ Your Mission:

* Set up 5 blocks of different health types
* Add 1 bird and fire at angle
* Log which blocks are destroyed after hit

📂 Submit: updated tools.py, classes.py, gameplay screenshot

**✅ Slide 10: Recap – From Code to Chaos**

✅ You now know how to:

* Detect collisions using Rect
* Damage blocks with precision
* Change visuals based on health

💡 Tip: Try using explosions or particles for advanced feedback later!

PPT 8

**Slide 1: Title Slide – “Detect. Damage. Destroy.”**

*Build Realistic Collisions and Breakable Structures in OMOTEC-PyGame: Siege of Avaria!*

* Introduce collision systems for dynamic gameplay.
* Inspired by Angry Birds, players launch birds to destroy fortresses.
* Focus: Detect impacts, apply damage, and update visuals.
* Learn to implement collisions using Pygame-CE.
* Enhances realism and engagement in two-player battles.

**Slide 2: What is Collision Detection?**

* Tracks when objects (e.g., birds, blocks, aliens) overlap or impact.
* Critical for scoring hits and triggering destruction in gameplay.
* Used to detect bird impacts on blocks or aliens.
* Without it, birds would pass through objects with no effect.
* Managed in **tools.py** for accurate game interactions.
* Key to realistic physics in Siege of Avaria!

**Slide 3: The Tools Behind Collisions**

* **tools.py**: Houses collision detection methods (e.g., check\_block\_collisions).
* **classes.py**: Defines Block class with take\_damage method and health.
* **assets.py**: Loads visuals like cracked block images or debris effects.
* **game\_modes.py**: Integrates collisions with scoring and win conditions.
* **media/**: Stores images and sounds for collision feedback.
* Collaborative modules ensure seamless damage system.

**Slide 4: Writing check\_block\_collisions()**

* In **tools.py**:
* def check\_block\_collisions(bird, blocks):
* for block in blocks:
* if bird.rect.colliderect(block.rect):
* block.take\_damage(bird.damage)
* Uses Pygame’s Rect.colliderect() for collision detection.
* Triggers take\_damage on block when bird hits.
* Supports both Quick and Basic modes.
* Updates game state for scoring and destruction.

**Slide 5: Adding take\_damage() to Block Class**

* In **classes.py**:
* class Block:
* def take\_damage(self, amount):
* self.health -= amount
* if self.health <= 0:
* self.destroy()
* Reduces block health based on bird’s damage value.
* Destroys block when health reaches zero.
* Interacts with **assets.py** for visual updates.
* Enables destructible environments in gameplay.

**Slide 6: Visual Feedback: Show Cracks or Debris**

* In **assets.py**: Load damaged block variants (e.g., cracked\_wood.png).
* In **tools.py**:
* if block.health < 50:
* screen.blit(cracked\_image, block.rect)
* Updates block sprite based on health level.
* Visual cues make damage intuitive for players.
* Images stored in **media/** for rendering.
* Enhances immersion in fortress destruction.

**Slide 7: Alien Collision Detection**

* In **tools.py**:
* def check\_alien\_collisions(bird, aliens):
* for alien in aliens:
* if bird.rect.colliderect(alien.rect):
* alien.destroy()
* return bonus\_score
* Removes alien on hit and awards bonus points.
* Triggers sound or animation via **assets.py**.
* Integrated in **game\_modes.py** for Quick Mode scoring.
* Adds variety to gameplay targets.

**Slide 8: Test and Debug Your System**

* **Test Case**:
  + Launch bird at a stone block.
  + Verify health reduces, sprite updates, and block disappears.
* Use print(block.health) in **tools.py** to track changes.
* Test in both Quick and Basic modes for consistency.
* Check **game\_modes.py** for score updates.
* Ensure visuals from **assets.py** render correctly.

**Slide 9: Student Task – Build a Damage-Test Arena**

* **Mission**:
  + Set up 5 blocks with different health (e.g., wood: 50, stone: 100).
  + Launch one bird at an angle to hit blocks.
  + Log which blocks are destroyed in console.
* **Deliverables**: Updated **tools.py**, **classes.py**, gameplay screenshot.
* Save as: my\_damage\_arena.py.
* Submit for feedback on collision accuracy.

**Slide 10: Recap – From Code to Chaos**

* **What you learned**:
  + Detect collisions using Pygame’s Rect.colliderect.
  + Apply damage with take\_damage in **classes.py**.
  + Update visuals based on health via **assets.py**.
  + Integrate collisions with scoring in **game\_modes.py**.
* **Design tip**: Add explosion or particle effects for advanced feedback.
* Reference: Pygame CE Documentation (<https://pyga.me/docs/>).

PPT 9

Based on the attached **OMOTEC-PyGame: Siege of Avaria! Report** and your training goals, here is a structured **10-slide student training presentation title outline** on:

**⚡ Power-Ups & Basic Game Strategy in PyGame**

*Train students to code, activate, and integrate power-up mechanics for strategic depth in gameplay.*

**✅ Slide 1: Title Slide – Supercharge Your Game**

*“Adding Power-Ups to Make Gameplay Smarter, Stronger, and Strategic”*

Introduce the topic and how power-ups are used in the Basic Game Mode of OMOGAME.

**✅ Slide 2: What Are Power-Ups?**

* Temporary **boosts** that affect gameplay
* Available in **Basic Mode only**
* Examples:
  + 🎯 **Double Damage**
  + 👁️ **Full Trajectory Preview**

📌 Designed to add excitement and tactics to bird launching.

**✅ Slide 3: Power-Up Module Map**

| **Module** | **Role in Power-Ups** |
| --- | --- |
| classes.py | Defines the PowerUp class & logic |
| game\_modes.py | Manages activation flow in Basic Game |
| assets.py | Loads icons & effects |
| media/ | Stores icons (e.g., lightning, radar) |

🧠 Think modular: each file plays a small, powerful part.

**✅ Slide 4: Creating the PowerUp Class**

class PowerUp:

def \_\_init\_\_(self, type, active=False):

self.type = type

self.active = active

* Add attributes: duration, cooldown, visual state
* Extend with methods like apply\_to\_bird(bird)

**✅ Slide 5: Activating Power-Ups in Basic Mode**

In game\_modes.py:

* Check power-up availability during player's turn
* Apply effects when bird is launched

if powerup.type == "double\_damage":

bird.damage \*= 2

💡 Use flags like powerup\_used = True to prevent re-use

**✅ Slide 6: Visualizing Power-Ups in the UI**

* Add power-up icons to the game screen
* Load with assets.py
* Display above HUD or next to player info

screen.blit(powerup\_icon, (x, y))

📦 Bonus: Flash effect when power-up is activated

**✅ Slide 7: Testing & Debugging Power-Ups**

🧪 Student steps:

* Trigger power-up
* Watch damage or trajectory behavior
* Use print() to verify activation:

print("Power-Up Applied:", powerup.type)

✔ One power-up at a time makes testing easier

**✅ Slide 8: Strategy Through Power-Ups**

Why power-ups matter in gameplay:

* Adds decision-making and **resource management**
* Encourages timing and planning  
  🧠 Tip: Don’t give both power-ups every turn!

📌 Optional: Let players earn or collect them

**✅ Slide 9: Student Challenge – Design & Code a New Power-Up**

Create a new power-up:

* Name it
* Code its effect
* Add a unique icon
* Test in a mock gameplay session

🎯 Examples: “Triple Shot”, “Freeze Alien”, “Reverse Gravity”

**✅ Slide 10: Recap – Coding with Strategy**

✅ What you’ve achieved:

* PowerUp class coded and applied
* Logic added to Basic Game
* Icons integrated into the screen
* Strategic gameplay decisions enabled

💡 Tip: Plan your power-up flow in a sketch before coding!

PPT 9

**Slide 1: Title Slide – Supercharge Your Game**

*“Adding Power-Ups to Make Gameplay Smarter, Stronger, and Strategic”*

* Introduce power-ups in OMOTEC-PyGame: Siege of Avaria!
* Enhance Basic Mode with strategic gameplay elements.
* Inspired by Angry Birds, players launch birds to destroy fortresses.
* Learn to implement power-ups using Pygame-CE.
* Focus: Add excitement and tactics to two-player battles.

**Slide 2: What Are Power-Ups?**

* Temporary boosts that alter gameplay dynamics.
* Exclusive to Basic Mode in Siege of Avaria!
* Examples:
  + 🎯 Double Damage: Doubles bird impact damage.
  + 👁️ Full Trajectory Preview: Shows complete bird flight path.
* Add excitement and strategic depth to bird launching.
* Encourage thoughtful decision-making in gameplay.
* Increase replayability with varied effects.

**Slide 3: Power-Up Module Map**

* **classes.py**: Defines PowerUp class with attributes and logic.
* **game\_modes.py**: Controls power-up activation in Basic Mode.
* **assets.py**: Loads power-up icons and visual effects.
* **media/**: Stores icons (e.g., lightningthink mode
* **tools.py**: Renders power-up visuals on screen.
* Modular design ensures seamless power-up integration.
* Each module plays a critical role in functionality.

**Slide 4: Creating the PowerUp Class**

* In **classes.py**:
* class PowerUp:
* def \_\_init\_\_(self, type, active=False):
* self.type = type
* self.active = active
* self.duration = 1

self.cooldown = 0

* Attributes: type, active, duration, cooldown.
* Method apply\_to\_bird(bird) modifies bird properties (e.g., damage).
* Supports power-ups like Double Damage and Trajectory Preview.
* Ensures temporary effects with controlled duration.

**Slide 5: Activating Power-Ups in Basic Mode**

* In **game\_modes.py**:
* if powerup.active and powerup.type == "double\_damage":
* bird.damage \*= 2

powerup\_used = True

* Check power-up availability during player’s turn.
* Apply effects when bird is launched.
* Use powerup\_used = True flag to prevent reuse.
* Reset after turn via **game\_modes.py** logic.
* Integrates with turn-based gameplay flow.

**Slide 6: Visualizing Power-Ups in the UI**

* Load icons in **assets.py**: pygame.image.load("media/double\_damage.png").
* Display icons above HUD or next to player info:

screen.blit(powerup\_icon, (x, y))

* Store icons in **media/** (e.g., lightning, radar).
* Bonus: Add flash effect on activation via **tools.py**.
* Enhances player feedback and UI clarity.
* Renders each frame for consistent visuals.

**Slide 7: Testing & Debugging Power-Ups**

* **Student steps**:
  + Trigger power-up in Basic Mode.
  + Observe changes in damage or trajectory behavior.
  + Verify with: print("Power-Up Applied:", powerup.type).
* Test one power-up at a time for clarity.
* Check **game\_modes.py** for activation logic.
* Ensure **assets.py** loads correct icons.
* Debug using console logs to track state.

**Slide 8: Strategy Through Power-Ups**

* Power-ups add decision-making and resource management.
* Encourage timing and planning for optimal use.
* Limit availability (e.g., one per turn) for balance.
* Optional: Players earn power-ups via gameplay actions.
* Enhance strategic depth in Basic Mode battles.
* Prevent overuse to maintain fair competition.
* Make every choice impactful for victory.

**Slide 9: Student Challenge – Design & Code a New Power-Up**

* **Task**:
  + Name a new power-up (e.g., “Triple Shot”, “Freeze Alien”, “Reverse Gravity”).
  + Code effect in **classes.py** and **game\_modes.py**.
  + Create unique icon in **media/** (e.g., triple\_shot.png).
  + Test in mock gameplay session via **main.py**.
* **Deliverables**: Updated code, screenshot of activation.
* Save as: my\_powerup.py.
* Submit for feedback on creativity and functionality.

**Slide 10: Recap – Coding with Strategy**

* **Achievements**:
  + Coded PowerUp class in **classes.py**.
  + Integrated logic into Basic Mode via **game\_modes.py**.
  + Added icons to UI with **assets.py** and **media/**.
  + Enabled strategic gameplay decisions.
* **Design tip**: Sketch power-up flow before coding.
* Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 10

Based on the report for **OMOTEC-PyGame: Siege of Avaria!** and the training goal of integrating scoring and game over logic, here's a **10-slide student training presentation title structure** focused on:

**🏁 Scoring System & Game Over Logic in PyGame**

*Learn how to calculate scores, track win conditions, and trigger the Game Over screen in OMOGAME*

**✅ Slide 1: Title Slide – “Track, Score, Win!”**

*“Build a scoring system that responds to real gameplay actions”*  
Introduce how scoring and game-over logic create structure and victory in games.

**✅ Slide 2: What is a Game State?**

* Keeps track of what's happening during a game
* Includes:
  + Current score
  + Whose turn it is
  + Whether someone has won  
    📌 Critical for both Quick and Basic Game modes

**✅ Slide 3: How Does Scoring Work in OMOGAME?**

* 🏆 Quick Game: First to reach **200 points** wins
* 💣 Basic Game: **Destroy all structures**
* Each hit adds points to a player’s score
* Aliens = bonus score

**✅ Slide 4: Score Tracking in Code**

In game\_modes.py, update player scores like:

player1\_score += hit\_value

🧪 Use print() to debug scoring logic

🧠 Tip: Use dictionaries for cleaner tracking:

scores = {1: 0, 2: 0}

**✅ Slide 5: Checking for Game Over**

🛑 Create a function:

def check\_game\_over():

if player1\_score >= 200:

return "Player 1 Wins!"

elif all\_blocks\_destroyed():

return "Player 2 Wins!"

💡 Called after each bird completes action

**✅ Slide 6: Displaying the Game Over Screen**

Use game\_screens.py to trigger end message:

show\_game\_over(winner)

🎉 Add restart and quit buttons  
🖼 Display final scores and victory badge

**✅ Slide 7: Updating the UI to Show Scores**

In main.py or game\_modes.py:

font = pygame.font.Font(None, 36)

text = font.render(f"Score: {scores[1]}", True, (255, 255, 255))

screen.blit(text, (50, 20))

📌 Live feedback makes gameplay feel rewarding

**✅ Slide 8: Testing Game Over Scenarios**

✅ Test Cases:

* Hitting multiple blocks to reach 200 points
* Destroying all blocks in Basic Mode
* Confirm score resets after restarting game

🛠️ Use logs: print("Checking Game Over...")

**✅ Slide 9: Student Task – Build Your Own Victory Rule**

🎯 Challenge:

* Add a **custom win condition**
* E.g., Alien defeated = instant win
* Add victory message: “Alien Slayer!”

📂 Deliverables:

* Updated game\_modes.py
* Screenshot of Game Over screen

**✅ Slide 10: Recap – Scoring = Structure**

✔ You now understand:

* Score tracking per player
* Detecting and triggering Game Over
* Displaying win results on screen

💬 Tip: Always test edge cases like tie scores or last-minute wins!

PPT 10

**Slide 1: Title Slide – “Track, Score, Win!”**

*“Build a scoring system that responds to real gameplay actions”*

* Introduce scoring and game-over logic in OMOTEC-PyGame: Siege of Avaria!
* Inspired by Angry Birds, players destroy fortresses for points.
* Learn to track scores and determine victory using Pygame-CE.
* Focus: Create structure and excitement in two-player battles.
* Covers both Quick and Basic Game modes.

**Slide 2: What is a Game State?**

* Tracks ongoing game dynamics and status.
* Includes:
  + Current score for each player.
  + Current player’s turn.
  + Win or loss conditions.
* Managed in **game\_modes.py** for both modes.
* Ensures fair gameplay and clear outcomes.
* Critical for responsive and structured gaming.

**Slide 3: How Does Scoring Work in OMOGAME?**

* **Quick Game**: First to 200 points wins via hits on blocks/aliens.
* **Basic Game**: Win by destroying all opponent structures.
* Points awarded per hit on blocks or aliens.
* Aliens provide bonus score for extra challenge.
* Scoring logic in **game\_modes.py** drives competition.
* Varies by mode for diverse gameplay experiences.

**Slide 4: Score Tracking in Code**

* In **game\_modes.py**:
* scores = {1: 0, 2: 0}

scores[1] += hit\_value

* Use dictionaries for clean score tracking.
* Update scores after collisions in **tools.py**.
* Debug with: print("Player 1 Score:", scores[1]).
* Supports both Quick and Basic mode scoring.
* Ensures accurate and fair point allocation.

**Slide 5: Checking for Game Over**

* In **game\_modes.py**:
* def check\_game\_over():
* if scores[1] >= 200:
* return "Player 1 Wins!"
* elif all\_blocks\_destroyed():

return "Player 2 Wins!"

* Called after each bird’s action completes.
* Checks Quick Mode (points) or Basic Mode (destruction).
* Triggers game-over screen via **game\_screens.py**.
* Ensures clear win/loss determination.

**Slide 6: Displaying the Game Over Screen**

* In **game\_screens.py**:
* def show\_game\_over(winner):

screen.blit(winner\_text, (x, y))

* Displays winner message and final scores.
* Adds restart and quit buttons for user control.
* Uses **assets.py** for victory badge images.
* Enhances player experience with clear feedback.
* Renders via **media/** resources for visuals.

**Slide 7: Updating the UI to Show Scores**

* In **main.py** or **game\_modes.py**:
* font = pygame.font.Font(None, 36)
* text = font.render(f"Score: {scores[1]}", True, (255, 255, 255))

screen.blit(text, (50, 20))

* Displays live scores for both players.
* Uses **assets.py** for font loading from **media/**.
* Provides rewarding feedback during gameplay.
* Updates each frame for real-time tracking.

**Slide 8: Testing Game Over Scenarios**

* **Test Cases**:
  + Hit multiple blocks to reach 200 points in Quick Mode.
  + Destroy all blocks in Basic Mode.
  + Verify score resets after game restart.
* Log checks: print("Checking Game Over...").
* Test in **game\_modes.py** for accuracy.
* Ensure **game\_screens.py** displays correct winner.

**Slide 9: Student Task – Build Your Own Victory Rule**

* **Challenge**:
  + Add custom win condition (e.g., “Alien defeated = instant win”).
  + Implement in **game\_modes.py**.
  + Show victory message: “Alien Slayer!” on game-over screen.
* **Deliverables**: Updated **game\_modes.py**, screenshot of game-over screen.
* Save as: my\_victory\_rule.py.
* Submit for feedback on logic and creativity.

**Slide 10: Recap – Scoring = Structure**

* **Achievements**:
  + Tracked scores per player in **game\_modes.py**.
  + Detected game-over conditions for both modes.
  + Displayed win results via **game\_screens.py**.
  + Enhanced UI with live score updates.
* **Debugging tip**: Test edge cases like ties or last-second wins.
* Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 11

Based on your objective and the attached **PYGAME LEVEL-3 Report**, here’s a structured **10-slide student training presentation** for:

**🎨 Asset Management & Audio Integration in PyGame**

*Train students to load images, organize assets, and experiment with sound using assets.py in OMOGAME.*

**✅ Slide 1: Title Slide – "Design What You Hear and See"**

*“Managing images, icons, and sounds to bring your game to life”*  
Introduce the role of **assets** in a game’s polish, performance, and player engagement.

**✅ Slide 2: What Are Game Assets?**

* 🖼️ Visuals: Birds, blocks, backgrounds, power-ups
* 🔊 Audio: Bird launch, crash sounds (optional with Pyodide)
* 🔤 Fonts: Game score, game over screens

📦 Stored in the **media/ folder**  
📌 Loaded centrally using **assets.py**

**✅ Slide 3: Why Use assets.py?**

* Keeps game files clean and modular
* Reduces memory loading errors
* Makes it easier to manage updates

🧠 Think of assets.py as your **game's media library manager**

**✅ Slide 4: Organize Your Media Folder**

📂 media/

* birds/: red.png, blue.png
* blocks/: stone.png, wood.png
* backgrounds/: battlefield.png
* fonts/: sci-fi.ttf
* icons/: powerup\_damage.png

📌 Use naming conventions for clarity

**✅ Slide 5: Load Assets in assets.py**

self.bird\_images = {

"red": pygame.image.load("media/birds/red.png"),

"blue": pygame.image.load("media/birds/blue.png")

}

self.font = pygame.font.Font("media/fonts/sci-fi.ttf", 36)

💡 Load all files once, use them everywhere

**✅ Slide 6: Add Backgrounds to the Game**

In main.py or game\_modes.py:

background = assets.background\_images["battlefield"]

screen.blit(background, (0, 0))

🎯 The right background sets the **theme and tone** for players

**✅ Slide 7: Integrate Audio (Optional)**

If using Pyodide-compatible audio:

import pygame.sndarray

import numpy as np

sound = pygame.sndarray.make\_sound(np.array([...]))

sound.play()

🎧 Sound on bird launch or collision adds **impact feedback**

**✅ Slide 8: Test for Load Performance**

* Use print("Loaded background") to verify asset access
* Check framerate: large assets can slow gameplay
* Test on different machines

🧪 Tip: Use compressed PNGs and MP3s for better speed

**✅ Slide 9: Student Task – Build a Themed Asset Set**

🎯 Your Challenge:

* Create your own visual theme (e.g., space, jungle)
* Replace:
  + Background
  + Bird icons
  + Block textures
* Load them in assets.py

📂 Deliver: updated media/ folder + preview screenshot

**✅ Slide 10: Recap – Build a Sensory Game World**

✅ You now understand how to:

* Organize and load assets
* Add backgrounds and fonts
* Experiment with simple sound logic  
  💬 Tip: Visual + audio feedback = more immersive gameplay

PPT 11

**Slide 1: Title Slide – "Design What You Hear and See"**

*“Managing images, icons, and sounds to bring your game to life”*

* Introduce asset management in OMOTEC-PyGame: Siege of Avaria!
* Inspired by Angry Birds, enhances two-player fortress battles.
* Focus: Use visuals and sounds for polish and engagement.
* Learn to manage assets with Pygame-CE for immersive gameplay.
* Covers assets for birds, blocks, and UI elements.

**Slide 2: What Are Game Assets?**

* **Visuals**: Birds (red.png), blocks (wood.png), backgrounds, power-up icons.
* **Audio**: Bird launch or crash sounds (optional with Pyodide).
* **Fonts**: Text for scores, game-over screens (e.g., sci-fi.ttf).
* Stored in **media/** folder for organized access.
* Loaded centrally via **assets.py** for efficiency.
* Critical for setting game tone and player immersion.

**Slide 3: Why Use assets.py?**

* Keeps game code modular and organized.
* Reduces memory errors by loading assets once.
* Simplifies updates to visuals or sounds.
* Acts as a media library manager for the game.
* Integrates with **main.py** and **game\_modes.py** for rendering.
* Ensures consistent asset use across Quick and Basic modes.

**Slide 4: Organize Your Media Folder**

* **media/** structure:
  + **birds/**: red.png, blue.png
  + **blocks/**: stone.png, wood.png
  + **backgrounds/**: battlefield.png
  + **fonts/**: sci-fi.ttf
  + **icons/**: powerup\_damage.png
* Use clear naming conventions (e.g., type\_name.png).
* Supports easy access in **assets.py** for loading.
* Keeps assets organized for scalable game design.

**Slide 5: Load Assets in assets.py**

* In **assets.py**:
* self.bird\_images = {
* "red": pygame.image.load("media/birds/red.png"),
* "blue": pygame.image.load("media/birds/blue.png")
* }

self.font = pygame.font.Font("media/fonts/sci-fi.ttf", 36)

* Load all assets once for efficient memory use.
* Supports rendering in **main.py** or **game\_modes.py**.
* Enables dynamic visuals for birds, blocks, and UI.
* Uses **media/** folder for centralized storage.

**Slide 6: Add Backgrounds to the Game**

* In **main.py** or **game\_modes.py**:
* background = assets.background\_images["battlefield"]

screen.blit(background, (0, 0))

* Sets game theme (e.g., battlefield, space) for immersion.
* Loaded via **assets.py** from **media/backgrounds/**.
* Rendered each frame in game loop for consistency.
* Enhances visual appeal in both game modes.
* Tested for smooth integration with gameplay.

**Slide 7: Integrate Audio (Optional)**

* For Pyodide-compatible audio in **assets.py**:
* import pygame.sndarray
* import numpy as np
* sound = pygame.sndarray.make\_sound(np.array([...]))

sound.play()

* Adds impact feedback for bird launches or collisions.
* Uses 2D NumPy arrays for stereo sound compatibility.
* Stored in **media/** (e.g., crash.mp3).
* Enhances immersion if supported by environment.

**Slide 8: Test for Load Performance**

* Verify asset loading: print("Loaded background") in **assets.py**.
* Check framerate to ensure large assets don’t slow gameplay.
* Test on different machines for consistent performance.
* Use compressed PNGs and MP3s for faster loading.
* Monitor **main.py** game loop for rendering issues.
* Debug with **tools.py** for visual rendering errors.

**Slide 9: Student Task – Build a Themed Asset Set**

* **Challenge**:
  + Create a visual theme (e.g., space, jungle).
  + Replace background, bird icons, and block textures.
  + Update **assets.py** to load new assets.
  + Capture screenshot of themed gameplay.
* **Deliverables**: Updated **media/** folder, **assets.py**, screenshot.
* Save as: my\_themed\_assets.py.
* Submit for feedback on theme and functionality.

**Slide 10: Recap – Build a Sensory Game World**

* **Achievements**:
  + Organized and loaded assets in **assets.py**.
  + Added backgrounds and fonts for visual polish.
  + Experimented with optional audio logic.
  + Created immersive gameplay with themed assets.
* **Design tip**: Prioritize compressed assets for performance.
* Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 12

Here’s a structured **10-slide student training presentation outline** focused on:

**🧩 Game Polish & Export Workflow in PyGame**

*Final step-by-step training to polish, package, and proudly present your OMOGAME project.*

**✅ Slide 1: Title Slide – “Final Touches & Game Packaging”**

*“Make your game shine, bug-free, and ready for submission!”*  
Introduce the final phase of development: **UI polish**, **testing**, and **project packaging**.

**✅ Slide 2: What Does Game Polish Mean?**

* Visual Refinement 🖌️ (e.g., spacing, alignment, colors)
* Consistent Menus & Navigation 🎮
* Bug Fixes 🐞
* Clean file structure 📂
* README + ZIP for sharing 📦

🧠 This is your game’s final impression—make it count!

**✅ Slide 3: Add Instruction & Credits Screens**

📘 In game\_screens.py, create:

* “How to Play” instructions
* Game mode details (Quick vs Basic)
* Author name, team role, credits

💡 Use buttons to navigate from menu → instruction → game

**✅ Slide 4: Refine Visual Layouts**

Check for:

* 🎯 Centered text and buttons
* ✅ Consistent fonts and sizes
* 🎨 Aligned power-up icons
* 🧱 No overlapping elements

📌 Use pygame.draw.rect() and padding values to tidy up layouts

**✅ Slide 5: Final Gameplay Testing Checklist**

🔄 Run full tests for:

* Bird Launching
* Block Destruction
* Alien Collisions
* Power-Up Activation
* Score Display
* Turn Switching
* Game Over Detection

💬 Use print logs: print("Collision detected") to trace issues

**✅ Slide 6: Create a README.md File**

📄 Content:

# OMOGAME – Siege of Avaria!

## How to Run:

1. Install Python

2. pip install pygame

3. Run: python main.py

## Game Modes:

- Quick: Score to 200

- Basic: Use power-ups to destroy

## Author:

Team [Name]

💡 Include a screenshot if possible!

**✅ Slide 7: Organize Your Project Folder**

Folder structure to submit:

FeatherFall/

│ main.py

│ game\_modes.py

│ classes.py

│ tools.py

│ assets.py

│ game\_screens.py

│ README.md

├── media/

│ ├── birds/

│ ├── blocks/

│ ├── backgrounds/

🧹 Remove unused files to keep it clean

**✅ Slide 8: Zip and Submit**

🗂️ Use:

* Right-click → Compress to ZIP (Windows/Mac)
* Online zip tools if needed  
  ✅ Ensure ZIP contains:
  + All .py files
  + media folder
  + README.md

📦 Final file: OMOGAME\_TeamName.zip

**✅ Slide 9: Class Presentation Tips**

🎤 In 5 minutes, show:

* The problem your game solves (fun strategy)
* Key mechanics (launching, scoring, power-ups)
* One exciting moment (alien hit, power-up use)
* Screenshots or a live demo  
  💡 Practice with a friend or mirror!

**✅ Slide 10: Final Recap – From Code to Completion**

✔ What you completed:

* UI polish + bug testing
* Instruction + credit screens
* Final packaging
* README.md
* Class-ready presentation

🎉 You built a fully working game from scratch—be proud and hit export!

PPT 12

**Slide 1: Title Slide – “Final Touches & Game Packaging”**

*“Make your game shine, bug-free, and ready for submission!”*

* Final phase for OMOTEC-PyGame: Siege of Avaria!
* Inspired by Angry Birds, a two-player strategic battle game.
* Focus: Polish UI, test thoroughly, and package for submission.
* Use Pygame-CE to ensure a professional, engaging experience.
* Prepare for a standout class presentation.

**Slide 2: What Does Game Polish Mean?**

* **Visual Refinement**: Align text, buttons, and power-up icons.
* **Consistent Navigation**: Smooth menu and screen transitions.
* **Bug Fixes**: Eliminate crashes or gameplay errors.
* **Clean Structure**: Organized files for easy sharing.
* **README + ZIP**: Clear instructions for running the game.
* Creates a lasting impression for players and evaluators.

**Slide 3: Add Instruction & Credits Screens**

* In **game\_screens.py**:
  + “How to Play” screen with gameplay basics.
  + Details for Quick (200 points) and Basic (power-ups) modes.
  + Credits: Author names, team roles, and asset sources.
* Add navigation buttons: Menu → Instructions → Game.
* Use **assets.py** for fonts and images from **media/**.
* Enhances user understanding and professionalism.

**Slide 4: Refine Visual Layouts**

* Ensure:
  + Centered text and buttons for clean UI.
  + Consistent fonts/sizes via **assets.py** (e.g., sci-fi.ttf).
  + Aligned power-up icons from **media/icons/**.
  + No overlapping elements in **game\_screens.py**.
* Use pygame.draw.rect() with padding for tidy layouts.
* Test visuals in both Quick and Basic modes.

**Slide 5: Final Gameplay Testing Checklist**

* Test key mechanics in **main.py** and **game\_modes.py**:
  + Bird launching (physics in **classes.py**).
  + Block destruction (collisions in **tools.py**).
  + Alien collisions (bonus points in **game\_modes.py**).
  + Power-up activation in Basic Mode.
  + Score display and turn switching.
  + Game-over detection in **game\_screens.py**.
* Debug with: print("Collision detected") in **tools.py**.

**Slide 6: Create a README.md File**

* **README.md** content:
* # OMOTEC-PyGame: Siege of Avaria!
* ## How to Run:
* 1. Install Python from https://www.python.org/downloads/
* 2. Run `pip install pygame`
* 3. Execute `python main.py`
* ## Game Modes:
* - Quick: First to 200 points by hitting blocks/aliens.
* - Basic: Destroy opponent’s structure with power-ups.
* ## Author:

Team [YourTeamName]

* Include a gameplay screenshot for visual appeal.
* Store in project root for clarity.

**Slide 7: Organize Your Project Folder**

* **OMOGAME FeatherFall/** structure:
  + main.py
  + game\_modes.py
  + classes.py
  + tools.py
  + assets.py
  + game\_screens.py
  + README.md
  + **media/**: birds/, blocks/, backgrounds/, fonts/, icons/
* Remove unused files for a clean submission.
* Verify all assets in **media/** are referenced in **assets.py**.

**Slide 8: Zip and Submit**

* Create ZIP:
  + Right-click **FeatherFall/** → Compress to ZIP.
  + Or use online tools for zipping.
* Ensure ZIP includes:
  + All .py files (main.py, game\_modes.py, etc.).
  + **media/** folder with all assets.
  + README.md with instructions.
* Name file: OMOGAME\_TeamName.zip.
* Test unzip to confirm all files are included.

**Slide 9: Class Presentation Tips**

* In 5-minute presentation:
  + Explain game’s purpose: Fun, strategic two-player battles.
  + Highlight mechanics: Bird launching, scoring, power-ups.
  + Showcase exciting moment (e.g., alien hit, power-up effect).
  + Share screenshots or run a live demo.
* Practice with a friend or mirror for confidence.
* Use **game\_screens.py** visuals for engaging slides.

**Slide 10: Final Recap – From Code to Completion**

* **Achievements**:
  + Polished UI with consistent layouts and navigation.
  + Thoroughly tested gameplay mechanics and bug fixes.
  + Added instruction and credits screens in **game\_screens.py**.
  + Packaged project with README.md and ZIP.
  + Prepared a compelling class presentation.
* **Tip**: Test edge cases (e.g., simultaneous wins) before submission.
* Reference: Pygame CE Documentation (https://pyga.me/docs/).

PPT 1 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**4. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 2 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 3 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 4 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 5 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 6 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 7 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 8 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 9 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 10 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 11 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**

PPT 12 EXERCISE

### **✅ Exercise: OMOGAME & PyGame-CE – Multiple Choice Questions**

**1. What is the primary objective of the Quick Mode in OMOGAME?**

A) Collect all power-ups before the opponent  
 B) Launch birds without any destruction  
 C) Reach a score of 200 points  
 D) Complete the entire storyline

✅ **Answer: C) Reach a score of 200 points**

**2. Which file is considered the main entry point of the OMOGAME project?**

A) tools.py  
 B) main.py  
 C) assets.py  
 D) classes.py

✅ **Answer: B) main.py**

**3. What does the tools.py file mainly handle in the game?**

A) Launching sound and animation  
 B) Player avatar selection  
 C) Collision detection and drawing functions  
 D) Managing game instructions

✅ **Answer: C) Collision detection and drawing functions**

**4. Which of the following is NOT listed as a unique feature of OMOGAME?**

A) Bird classes with special physics  
 B) Alien targets for bonus points  
 C) Multiplayer online matchmaking  
 D) Destructible block environments

✅ **Answer: C) Multiplayer online matchmaking**

**5. Why is organizing the OMOGAME project into files like classes.py, media/, and game\_modes.py important?**

A) It reduces the game’s memory usage  
 B) It helps reduce the number of bugs completely  
 C) It makes the project look colorful  
 D) It simplifies managing and updating different game parts

✅ **Answer: D) It simplifies managing and updating different game parts**