A Unity Developer's Guide to 3D Chess

Part 1: Setting the Stage: Project, Scene, and Camera

This foundational section of the textbook is dedicated to establishing a robust and professional project environment within the Unity Editor. Before any game logic is written or a single chess piece is moved, a well-structured project is paramount for a smooth and scalable development process. The following chapters will guide the developer through creating a new Unity project using modern rendering pipelines, organizing project assets efficiently, constructing the 3D game world, configuring a dynamic and user-friendly camera system, and establishing a visually compelling lighting scheme. These initial steps are crucial, as they form the canvas upon which the entire game will be built.

Chapter 1: Creating Your First Unity Project

The journey begins in the Unity Hub, the central application for managing Unity Editor installations and projects. A proper project setup from the outset ensures compatibility, performance, and organization.

1.1 Unity Hub and Editor Versioning

First, it is necessary to launch the Unity Hub. For a new project, selecting an appropriate Editor version is a critical decision. Unity offers different release tracks, but for most projects, a **Long Term Support (LTS)** version is the recommended choice.[1] LTS versions receive continued support and bug fixes for an extended period, providing a stable foundation for development, which is especially important for a project with the complexity of a chess game.

1.2 Selecting the 3D (URP) Template

When creating a new project, Unity presents several templates. For this 3D chess game, the **3D (URP)** template is the ideal starting point.[2] URP, or the Universal Render Pipeline, is Unity's modern, scriptable rendering solution. It is optimized for performance across a wide range of platforms, from mobile devices to high-end PCs, and provides access to advanced graphical tools like the Shader Graph and post-processing effects that are integrated directly into the pipeline.[1]

The creation process is as follows:

1. In the Unity Hub, navigate to the "Projects" tab and click the "New project" button.
2. Select the desired LTS Editor version from the dropdown menu.
3. From the list of templates, select "3D (URP)".
4. Assign a project name, such as "Unity3DChess," and choose a location on the local drive.
5. Click "Create project." Unity will then assemble the new project, which may take several minutes.

1.3 Professional Folder Structure

Once the project opens in the Unity Editor, the first task is to organize the Assets folder. A clean folder structure is a hallmark of professional game development. It makes assets easier to find, manage, and debug, which becomes increasingly important as the project grows in complexity.

Within the Assets folder in the Project window, create the following subfolders [3]:

- _Scenes: To store all game scenes, such as the main gameplay scene and any menu scenes.
- _Scripts: This will house all C# scripts.
- _Prefabs: For storing reusable GameObjects, such as the chess pieces and board tiles.
- _Materials: For all the materials that define the appearance of the 3D models.
- _Models: To store the 3D model files (e.g., FBX, OBJ) for the chessboard and pieces.
- _Textures: For all image files used by materials, such as wood grain or marble patterns.

This organized structure prevents the root Assets directory from becoming cluttered

and ensures that any developer joining the project can quickly understand its layout.

Chapter 2: Building the 3D Environment and Chessboard

With the project structure in place, the next step is to create the 3D world. This involves setting up a basic ground plane and, most importantly, constructing the 8x8 chessboard.

2.1 Creating the Ground Plane

A simple ground plane provides a surface for the chessboard to rest on and helps ground the scene visually.

1. In the Hierarchy window, go to GameObject > 3D Object > Plane.
2. Rename this new GameObject to "Ground".
3. In the Inspector, set its Transform position to (0, 0, 0) and scale it up to provide an ample floor area, for example, (10, 1, 10). This plane will catch shadows and add a sense of place to the game.[5]

2.2 Approaches to Chessboard Creation

There are two primary methods for creating the chessboard itself.

1. **Single Mesh with a Texture:** This approach uses a single 3D model for the entire board. The checkerboard pattern is applied as a texture image. This method is generally more performant as it results in fewer GameObjects and draw calls. However, it requires knowledge of UV mapping in a 3D modeling program and makes interacting with individual squares via code more complex, as it requires translating a raycast hit point on a single mesh into grid coordinates.[6]
2. **Procedural Generation with Individual Tiles:** This method involves programmatically creating 64 individual GameObjects, one for each square of the board, using a C# script. Each tile is a separate object with its own collider. This approach offers maximum flexibility for game logic, as each tile can be easily referenced, highlighted, or queried for its contents.[7]

For this textbook, the procedural generation method will be used. It provides a clearer and more direct foundation for learning the core game logic of selecting and moving pieces between discrete tiles.

2.3 Procedural Board Generation

To begin, a "blueprint" or prefab for a single board tile is needed.

1. Create a tile GameObject by navigating to GameObject > 3D Object > Quad. A quad is a simple, one-sided plane, which is more efficient than a six-sided cube for this purpose.
2. Rename it "Tile" and ensure its Transform position is at (0, 0, 0).
3. This "Tile" GameObject will be turned into a prefab later, which the board generation script will use to instantiate all 64 squares.

Chapter 3: Configuring the Top-Down Camera

The camera is the player's window into the game world. For a chess game, a clear, top-down, slightly angled view is ideal, similar to what is found in many real-time strategy (RTS) games.

3.1 Initial Camera Placement and Projection

The default Main Camera in the scene needs to be positioned correctly. A good starting point for its Transform values is:

- **Position:** X: 0, Y: 10, Z: -7
- **Rotation:** X: 55, Y: 0, Z: 0

These values place the camera above the center of the board, looking down at an angle.[10] This provides a clear view of all the squares while maintaining a sense of three-dimensional depth.

Next, consider the camera's **Projection** mode. This can be set in the Camera component in the Inspector.

- **Perspective:** This mode mimics human vision, where objects farther away appear smaller. It provides a more realistic and cinematic look.
- **Orthographic:** This mode removes all perspective, rendering all objects at their true size regardless of distance. It offers excellent clarity for strategy games but can look flat and less visually engaging for a 3D game.[11]

For this project, a **Perspective** camera will be used to enhance the 3D aesthetic. The **Field of View** can be adjusted to control the amount of zoom and distortion; a value

around 40-50 is typically suitable.

3.2 Implementing a Camera Rig for Flexible Control

A common mistake for beginners is to apply movement scripts directly to the Main Camera GameObject. A more robust and professional technique is to create a "camera rig." This involves parenting the camera to an empty GameObject, which then handles all movement and rotation. This separation of concerns simplifies the control scripts significantly.[10]

1. Create a new empty GameObject via GameObject > Create Empty.
2. Name it "CameraRig".
3. Reset its Transform to (0, 0, 0).
4. Drag the Main Camera GameObject onto the CameraRig in the Hierarchy to make it a child object.

Now, any scripts for panning the camera will be attached to the CameraRig, moving both the parent and child camera together. Scripts for zooming can be attached to the Main Camera itself, moving it along its local forward axis without affecting the rig's position. This decoupling makes the code for each function cleaner and more maintainable.

3.3 Scripting an RTS-Style Camera Controller

To give the player control, a simple C# script can be created to handle panning and zooming.

1. Create a new C# script named CameraController inside the _Scripts folder.
2. Attach this script to the CameraRig GameObject.

This script will read keyboard input (WASD or arrow keys) to move the CameraRig on the XZ plane (panning) and mouse scroll wheel input to move the Main Camera forward and backward along its local Z-axis (zooming).[10] This provides a smooth, intuitive way for the player to adjust their view of the game.

Chapter 4: Fundamentals of 3D Lighting

Lighting is one of the most powerful tools for establishing the mood and visual quality of a scene. A well-lit scene can elevate a simple project to look polished and

professional.

4.1 The Three-Point Lighting Model

A fundamental technique in cinematography and computer graphics is three-point lighting. It uses three light sources to illuminate a subject (in this case, the chessboard and pieces) in a way that creates depth and visual interest.[15]

- **Key Light:** The primary and brightest light source, which establishes the main illumination and casts the primary shadows. The default Directional Light in a new Unity scene can serve as the key light.
- **Fill Light:** A secondary, less intense light positioned opposite the key light. Its purpose is to "fill in" the harsh shadows created by the key light, revealing detail in the darker areas.
- **Rim Light (or Back Light):** A light placed behind the subject. It creates a bright outline or "rim" around the object, helping to separate it from the background and add a sense of depth.

4.2 Setting Up the Scene Lights

1. **Key Light (Directional Light):** Select the default Directional Light in the Hierarchy. Adjust its Rotation so that it is not pointing straight down, but at an angle. This will create more interesting, elongated shadows that reveal the 3D form of the chess pieces.[16] A slightly warm, yellowish
   Color can simulate sunlight.
2. **Fill Light (Point Light):** Create a Point Light via GameObject > Light > Point Light. Position it on the opposite side of the board from the Directional Light's angle. Set its Intensity to a lower value (e.g., 0.5) and its Color to a cool, subtle blue to contrast with the warm key light. This subtle use of complementary colors adds significant visual appeal.[5]
3. **Rim Light (Spot Light):** A Spot Light can be used effectively as a rim light. Create one via GameObject > Light > Spot Light and position it behind the board, angled toward the pieces. This will create a highlight on the back edges of the pieces, making them "pop" from the board.[15]

4.3 Baked vs. Real-time Lighting

Unity provides two primary modes for calculating lighting:

- **Real-time Lighting:** Lights and shadows are calculated every frame. This is necessary for dynamic objects that move, but it is computationally expensive.[18]
- **Baked Lighting (Global Illumination):** For static objects that do not move, Unity

can pre-calculate complex lighting interactions, including how light bounces off surfaces (indirect lighting) and soft shadows. This process, called "baking," stores the lighting information in texture maps (lightmaps). The result is significantly higher visual quality at a much lower performance cost during gameplay.[19]

Since the chessboard and the ground plane are static, they should be marked for lightmap baking.

1. Select the "Ground" and "Board" GameObjects.
2. In the Inspector, check the "Static" checkbox in the top-right corner.[5]
3. Open the Lighting window (Window > Rendering > Lighting).
4. Navigate to the "Scene" tab and click the "Generate Lighting" button. Unity will begin the baking process.

4.4 Enhancing Visuals with Post-Processing

Post-processing effects are filters applied to the camera's final image, much like photo editing filters. They can dramatically improve the final look of the game. With URP, these are managed through a Global Volume.

1. Create a Global Volume via GameObject > Volume > Global Volume.
2. Create a new Volume Profile by clicking the "New" button in the Volume component.
3. Add overrides for simple yet effective effects [21]:
    - **Bloom:** This effect makes bright areas of the scene bleed light, giving a soft, glowing appearance. It can add a touch of magic or high quality to the lighting.
    - **Vignette:** This darkens the corners of the screen, which helps to focus the player's attention on the center of the action—the chessboard.

By completing these foundational steps, the project is now set up with a professional structure, a functional camera, and an appealing visual base, ready for the introduction of the game's primary assets.

Part 2: The Pieces and the Board: Assets and Materials

This section focuses on populating the scene with its essential actors: the chess pieces and the board. It covers the complete asset pipeline, from acquiring 3D models

to defining their surface appearance with materials and textures in Unity. A developer can choose to use pre-made assets to accelerate development or engage in the rewarding process of modeling their own unique set.

Chapter 5: Sourcing and Importing Free 3D Chess Assets

For developers eager to focus on the programming and game logic, using pre-made 3D assets is the most efficient path. The internet offers a wealth of high-quality models, many of which are available for free.

5.1 Finding Game-Ready Assets

Several online marketplaces are excellent sources for 3D models. When searching, it is beneficial to use terms like "chess set," "low poly," and "game ready" to find models optimized for real-time performance. Reputable sources include:

- **TurboSquid** [22]
- **CGTrader** [24]
- **Clara.io** [26]
- **Free3D** [27]
- **Unity Asset Store** [1]

5.2 Understanding File Formats and Licensing

When downloading assets, two key factors must be considered:

- **File Format:** Unity has robust support for the .FBX format, which is the industry standard for game assets as it can contain mesh data, materials, textures, and animations. The .OBJ format is also widely supported but is typically limited to mesh data. Unity can also directly import proprietary files from 3D modeling software like Blender (.blend) or Maya (.ma), provided the source application is installed on the same machine.[29] For simplicity and compatibility, exporting or downloading assets as
  .FBX is the recommended practice.[29]
- **Licensing:** This is a critical consideration. "Free" does not always mean free for any use. Each asset will have a specific license (e.g., Creative Commons, Royalty-Free, Editorial Use). It is imperative to read and understand the license terms to ensure the asset can be used in a personal, educational, or commercial

project without violating the creator's rights.[22]

5.3 Importing and Configuring Models in Unity

Once a suitable chess set has been downloaded and unzipped, the import process into Unity is straightforward:

1. Drag the model files (e.g., King.fbx, Queen.fbx) and any associated texture images into the _Models and _Textures folders in the Unity Project window, respectively.[30]
2. Select an imported model file in the Project window to open its **Import Settings** in the Inspector.[29]
3. Key settings to review include:
    - **Scale Factor:** This can be used to adjust the size of the model on import if it appears too large or small.
    - **Materials Tab:** This tab controls how Unity handles materials embedded in the model file. The "Use Embedded Materials" option can be used to extract them, or materials can be created manually in Unity for greater control.[29]

After configuring the settings, clicking "Apply" will finalize the import process. The models can then be dragged from the Project window into the Scene view to be viewed.

Chapter 6: (Optional) Modeling Your Own Chess Set in Blender

For those seeking a more custom look or wishing to develop their 3D modeling skills, creating a unique chess set is a rewarding challenge. Blender is a powerful, free, and open-source 3D creation suite that is perfectly suited for this task.[33]

6.1 Modeling a Rook: A Step-by-Step Example

The rook is an excellent piece to start with, as its shape is based on simple geometric extrusion. The following steps outline the process [33]:

1. **Setup:** In Blender, delete the default cube. Add a Circle to the scene (Add > Mesh > Circle). This will form the base of the rook.
2. **Extrusion:** Enter Edit Mode (Tab key). With the circle's vertices selected, use the **Extrude** tool (E key) to pull the shape upwards along the Z-axis.
3. **Scaling:** After extruding a section, use the **Scale** tool (S key) to widen or narrow the profile, creating the rook's curved shape. Repeat the Extrude-Scale process

to build the entire profile of the rook's body.

4. **Refinement:** To create a smooth, high-quality surface, add a **Subdivision Surface Modifier** from the Modifiers tab. This will subdivide the mesh, creating a more rounded appearance. To create sharp edges and creases, use the **Loop Cut and Slide** tool (Ctrl+R) to add edge loops close to the areas that need to be sharpened.
5. **Creating the Battlements:** To cut the notches into the top of the rook, the **Boolean Modifier** is used. Create a Cube and position it so it intersects with the top of the rook where a notch should be. Select the rook, add a Boolean Modifier, set the operation to "Difference," and select the cube as the target object. This will subtract the cube's volume from the rook's mesh. This process can be repeated for all notches.
6. **Exporting:** Once the model is complete, export it as an .FBX file (File > Export > FBX (.fbx)). This format is ideal for importing into Unity.[30]

The other pieces can be created using similar techniques. The pawn, bishop, queen, and king are all "lathe" objects that can be modeled by extruding and scaling a circle. The knight is the most complex piece and often requires more advanced box modeling or sculpting techniques to create its distinct horse-head shape.[34]

Chapter 7: Creating and Applying Materials for the Board and Pieces

Materials define the surface properties of a 3D object—its color, texture, shininess, and roughness. In Unity, materials are separate assets that are applied to the Mesh Renderer component of a GameObject.

7.1 Creating the Core Materials

For the chess game, at least four distinct materials are needed: one for the light squares, one for the dark squares, one for the white pieces, and one for the black pieces.

1. In the _Materials folder in the Project window, right-click and select Create > Material.[35]
2. Create four materials and name them appropriately: Board_Light, Board_Dark, Piece_White, and Piece_Black.

7.2 Configuring Material Properties

Selecting a material asset will display its properties in the Inspector. The URP Lit shader has several key parameters:

- **Base Map:** This property, also known as Albedo, determines the fundamental color of the material. A solid color can be selected, or an image texture (like a wood grain or marble pattern) can be assigned.[35] For example, Board_Light could be a light wood texture, and Board_Dark could be a dark wood texture.
- **Metallic:** This slider controls how much the material behaves like a metal. A value of 0 is non-metallic (dielectric), while 1 is fully metallic.
- **Smoothness:** This slider controls the microsurface scattering of the material. A high value creates a smooth, glossy surface that reflects light clearly (like polished marble), while a low value creates a rough, matte surface that diffuses light (like unfinished wood).[36]

For the chess set, one might create a Piece_White material with a white base color, a high smoothness value, and a slight metallic value to give it a polished look.

7.3 Applying Materials Programmatically

The materials for the board tiles and chess pieces will be assigned via C# scripts during the setup phase.

- **Board Materials:** The script that procedurally generates the 64 board tiles will determine if a tile is light or dark based on its grid coordinates. It will then assign either the Board_Light or Board_Dark material to the tile's Mesh Renderer component.[8]
- **Piece Materials:** The script that spawns the chess pieces will assign the Piece_White or Piece_Black material based on which player the piece belongs to.[8]

A crucial performance optimization relates to how these materials are assigned. When a script accesses renderer.material, Unity creates a unique copy (an instance) of that material for that specific object. If this is done for all 32 light squares, it results in 32 separate draw calls. A more efficient method is to use renderer.sharedMaterial. This modifies the source material asset directly and allows all objects using it to be rendered in a single batch, significantly improving performance.[38] For the board tiles, which do not need unique properties,

sharedMaterial is the superior choice. For the pieces, where one might want to

highlight a selected piece by changing its material properties at runtime, using .material is necessary to avoid changing the appearance of all pieces of that color simultaneously.

Part 3: The Brains of the Game: Core Architecture and Game State

This section delves into the heart of the project: the C# code that will govern the game's logic and state. A well-designed software architecture is essential for managing the complexity of chess rules. The focus will be on the principle of **separation of concerns**, ensuring that the game's underlying logic is independent of its visual presentation. This creates a codebase that is easier to debug, maintain, and extend.

Chapter 8: Designing the Game's Architecture

A robust architecture is the blueprint for the game's code. It defines how different systems communicate and what responsibilities each component has.

8.1 The Principle of Separation of Concerns

A core tenet of good software design is to separate the data model (the state of the game) from the view (how the game is displayed) and the controller (the logic that handles player input and updates the model).[39]

- **Model:** The abstract representation of the game. This includes the state of the 8x8 board, the position of each piece, the current player's turn, and all the rules of chess. This part of the code should have no knowledge of Unity's GameObjects, 3D rendering, or UI.
- **View:** The visual representation of the model. This consists of the 3D GameObjects for the board and pieces, the materials, the lighting, and the UI elements. The view's job is to accurately reflect the current state of the model.
- **Controller:** The bridge between the model and the view. It listens for player input (like mouse clicks), translates that input into commands for the model (e.g., "move

piece from E2 to E4"), and then tells the view to update itself based on the model's new state.

This separation makes the system modular. One could, for example, replace the entire 3D view with a 2D one without having to change any of the core chess logic in the model.

8.2 Defining the Key C# Scripts

Based on this architecture, the project will be built around a few key C# scripts:

- GameManager.cs: This will be a central singleton class that manages the overall game flow. It will track the current player, the game state (e.g., in-progress, checkmate, stalemate), and handle turn switching.[42]
- Board.cs: This class will be responsible for both the logical representation of the board (the data structure holding the pieces) and the visual representation (managing the tile and piece GameObjects).
- Piece.cs: An abstract base class that defines the common properties and behaviors for all chess pieces, such as their color and an abstract method for generating moves.[8]
- Pawn.cs, Rook.cs, Knight.cs, etc.: These are concrete subclasses that inherit from Piece.cs. Each class will implement the unique movement logic for that specific piece type.[42]
- InputController.cs (formerly TileSelector.cs): This script will be dedicated to handling player input. It will use raycasting to detect clicks on the board and will communicate the player's intended actions to the GameManager.[45]

Chapter 9: Representing the Board: Data Structures in C#

The logical representation of the chessboard is the foundational data structure of the entire game logic. Its efficiency and clarity are paramount.

9.1 The Two-Dimensional Array

For a grid-based game like chess, the most intuitive and effective data structure is a two-dimensional array.[8] This structure directly maps to the 8x8 grid of the board. The array will be declared in the

Board.cs script:

C#

```
private Piece[,] pieces = new Piece;
```

Each element in this array will hold a reference to the Piece script instance that occupies the corresponding square on the board. If a square is empty, its corresponding array element will be null. This allows for constant-time lookup of any square's contents, which is highly efficient.

While advanced chess engines often use a more complex data structure called a "bitboard" for maximum performance, a 2D array is perfectly sufficient for this project and is far more readable and easier to work with for learning purposes.[48]

9.2 Establishing a Coordinate System

A consistent coordinate system is crucial to avoid bugs. The convention will be to map the array indices to standard algebraic chess notation as follows:

- The array index `` will correspond to square **a1**.
- The array index `` will correspond to square **h1**.
- The array index `` will correspond to square **a8**.
- The array index `` will correspond to square **h8**.

Essentially, the first index represents the file (column, 0-7 for a-h) and the second index represents the rank (row, 0-7 for 1-8).

Chapter 10: Scripting the Chess Pieces and Spawning the Board

This chapter focuses on creating the C# classes for the pieces and writing the initialization logic that sets up the board at the start of the game.

10.1 The Piece Base Class

An abstract base class is used to define the shared characteristics of all pieces,

promoting code reuse.

1. Create a new C# script named Piece.cs.
2. Define an enum for the player colors: public enum PlayerColor { White, Black }.
3. The Piece class will be abstract and inherit from MonoBehaviour:

C#

```
public abstract class Piece : MonoBehaviour
{
    public PlayerColor color;
    public abstract List<Vector2Int> GetAvailableMoves(Vector2Int currentPos, Board board);
}
```

The GetAvailableMoves method is abstract, meaning each concrete piece subclass (Pawn, Rook, etc.) *must* provide its own implementation of this method, which is where the unique movement rules will reside.[8]

10.2 Piece Subclasses

For each type of chess piece, a new script will be created that inherits from Piece.cs. For example, Rook.cs:

C#

```
public class Rook : Piece
{
    public override List<Vector2Int> GetAvailableMoves(Vector2Int currentPos, Board board)
    {
        // Logic for Rook movement will be implemented here.
        List<Vector2Int> moves = new List<Vector2Int>();
        //...
        return moves;
    }
}
```

10.3 Board Setup and Piece Spawning

The Board.cs script will contain the methods to create the visual board and place the pieces in their starting positions.

1. **Create Prefabs:** Each 3D model for a chess piece should be turned into a prefab. Create an empty GameObject, add the 3D model as a child, attach the corresponding piece script (e.g., Rook.cs), and drag it into the _Prefabs folder.
2. **Board Generation:** The CreateBoard() method will use a nested for loop to instantiate 64 Tile prefabs, arranging them in an 8x8 grid and assigning the light and dark materials appropriately.[8]
3. **Piece Spawning:** A SetupPieces() method will be responsible for placing the pieces. It will iterate through the pieces array and, based on the standard chess starting layout, instantiate the correct piece prefab at each required position. It will also set the piece's color and assign the correct material (Piece_White or Piece_Black).[8]

The following table provides a clear reference for the initial board setup, mapping the standard notation to the array indices that will be used in the SetupPieces() method.

Piece	Color	Standard Notation	Array Index [col, row]
Rook	White	a1	``
Knight	White	b1	``
Bishop	White	c1	``
Queen	White	d1	``
King	White	e1	``
Bishop	White	f1	``
Knight	White	g1	``
Rook	White	h1	``
Pawn	White	a2-h2	[0-7, 1]

Pawn	Black	a7-h7	[0-7, 6]
Rook	Black	a8	``
Knight	Black	b8	``
Bishop	Black	c8	``
Queen	Black	d8	``
King	Black	e8	``
Bishop	Black	f8	``
Knight	Black	g8	``
Rook	Black	h8	``

This structured approach ensures that the game starts with a correctly configured logical state and a perfectly matching visual representation.

Part 4: The Player's Hand: Input and Interaction

This section focuses on creating the crucial link between the player and the game's logic. It covers the implementation of systems that detect player input, translate it into meaningful game actions, and provide clear visual feedback to guide the player's decisions. This interactive loop is fundamental to creating an engaging and intuitive gameplay experience.

Chapter 11: Selecting Pieces with Mouse Clicks and Raycasting

To allow a player to interact with the 3D board, the game must be able to determine what object is under the mouse cursor when a click occurs. The standard technique

for this in 3D Unity projects is **raycasting**.

11.1 The Fundamentals of Raycasting

Raycasting is the process of sending out an invisible ray from a point in a specific direction to detect any colliders it intersects with.[51] For a mouse-based selection system, a ray is cast from the camera's position, through the point on the screen where the mouse cursor is located, and into the 3D scene.[51]

To make objects detectable by a raycast, they must have a Collider component attached. Both the tile prefabs and the piece prefabs must be configured with a Box Collider or Mesh Collider.

11.2 Implementing the Input Controller

The InputController.cs script will manage this process. Its primary logic will reside in the Update() method, which is called once per frame.

1. **Detecting a Click:** The script first checks if the left mouse button has been pressed using Input.GetMouseButtonDown(0).
2. **Creating the Ray:** When a click is detected, a ray is created from the camera:
   C#
   ```
   Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
   ```

3. **Casting the Ray:** The Physics.Raycast() method is then used to perform the cast. It returns true if it hits a collider and provides detailed information about the hit via a RaycastHit object.
   C#
   ```
   if (Physics.Raycast(ray, out RaycastHit hit))
   {
       // A collider was hit. Process the hit object.
       GameObject hitObject = hit.collider.gameObject;
       //... logic to identify if it's a tile or piece
   }
   ```

4. **Identifying the Target:** The script then needs to determine what was clicked. This can be done by checking the tag of the hitObject or by attempting to get a specific component from it, such as a Tile or Piece script.
5. **Communicating with the GameManager:** Once a tile or piece is identified, the InputController will call a public method on the GameManager singleton, passing along the relevant information (e.g., the grid coordinates of the clicked tile or a

reference to the selected piece).

Chapter 12: Providing Visual Feedback: Highlighting Selections and Legal Moves

Effective visual feedback is essential for a good user experience. The player needs to know which piece is currently selected and what their valid move options are.

12.1 Highlighting the Selected Piece

When the GameManager confirms that a valid piece has been selected, it needs to trigger a visual change. There are several ways to achieve this:

- **Material Swapping:** The simplest method is to change the piece's material to a distinct "highlight" material that might be brighter or have an emissive glow.[53]
- **Using a Projector or Decal:** A more sophisticated approach involves instantiating a GameObject with a Projector component that projects a circle or other indicator onto the tile beneath the selected piece.
- **Outline Shader:** An advanced technique involves using a custom shader to render an outline around the selected piece's mesh.[54]

For this textbook, a simple material property change will be used. When a piece is selected, its material instance will have its color slightly brightened. When deselected, it will be returned to its original color.

12.2 Displaying Legal Moves

Once a piece is selected, the game must show the player where it can legally move.

1. The GameManager receives the selected piece from the InputController.
2. It calls that piece's GetAvailableMoves() method to get a list of valid target coordinates (Vector2Int).
3. For each valid coordinate in the returned list, the GameManager will instantiate a "move highlight" prefab onto the corresponding tile. This prefab can be a simple, semi-transparent 3D cylinder or a quad with a green circle texture.[55] These highlights visually communicate the player's options.
4. When the player either makes a move, selects a different piece, or deselects the current piece, these highlight GameObjects must be destroyed to clean up the board.[55]

12.3 A State-Driven Approach to Interaction

The player's interaction with the board can be modeled as a simple **state machine**. This architectural pattern makes the input handling logic much cleaner and more robust than a single, monolithic Update loop filled with complex conditional statements.[45] The game will primarily be in one of two states:

- **State 1: Awaiting Piece Selection:**
  - In this state, the InputController is listening for clicks.
  - A click on a piece belonging to the current player will trigger a transition. The GameManager will be notified, the piece will be highlighted, its legal moves will be calculated and displayed, and the game will transition to the next state.
  - Clicks on empty tiles or enemy pieces are ignored.
- **State 2: Piece Selected, Awaiting Move:**
  - In this state, a piece is already selected and its moves are highlighted.
  - A click on a highlighted tile signifies a move. The InputController informs the GameManager, which will execute the move, update the board state, and transition back to State 1 (after switching the current player).
  - A click on another friendly piece will switch the selection, keeping the game in State 2 but updating the highlighted piece and moves.
  - A click anywhere else (an invalid tile or the same piece) will deselect the piece, clear the move highlights, and transition the game back to State 1.

This state-based design ensures that player input is interpreted correctly based on the current context of the game, leading to a more predictable and bug-free system.

Part 5: Mastering the Rules: Implementing Chess Logic in C#

This section is the programming core of the project, where the abstract rules of chess are translated into concrete C# code. The object-oriented approach established earlier, with a base Piece class and specific subclasses for each piece type, provides an elegant framework for encapsulating the unique movement logic. Each chapter will build upon this foundation, progressively implementing the full set of chess rules, from basic movement to the game's complex special maneuvers.

Chapter 13: Basic Movement: Pawns and Knights

The implementation begins with two of the most unique pieces in terms of movement.

13.1 The Pawn

The pawn's movement is the most conditional. Its GetAvailableMoves implementation must account for several rules:

- **Forward Direction:** The definition of "forward" depends on the pawn's color. A white pawn on rank 2 (array index 1) moves towards rank 8 (array index 7), while a black pawn moves in the opposite direction. The code must check piece.color to determine the correct direction vector (e.g., (0, 1) for white, (0, -1) for black).
- **Initial Two-Square Move:** A pawn that has not yet moved can advance two squares forward, provided the path is clear. A bool hasMoved flag in the Piece base class will be used to track this.
- **Standard One-Square Move:** A pawn can always move one square forward to an empty tile.
- **Diagonal Capture:** A pawn can capture an opponent's piece on a diagonally adjacent square. The logic must check if the target diagonal squares are occupied by an enemy piece.

13.2 The Knight

The knight's move is simpler to implement because it is the only piece that can "jump" over other pieces, meaning its path does not need to be checked for obstructions.[56] Its

GetAvailableMoves method will:

1. Define an array of eight Vector2Int offsets that represent all possible L-shaped moves: (1, 2), (2, 1), (-1, 2), (-2, 1), etc.
2. Iterate through these eight offsets.
3. For each offset, calculate the potential target square.
4. Check if the target square is within the bounds of the 8x8 board.
5. Check if the target square is either empty or occupied by an enemy piece. If so, it is a valid move.

Chapter 14: Sliding Pieces: Rooks, Bishops, and the Queen

The rook, bishop, and queen are "sliding" pieces, meaning they can move any number of squares along a line until they are blocked.

14.1 Rooks and Bishops

The logic for these pieces involves iterating along specific directions.

- **Direction Vectors:** A set of direction vectors will be defined for each piece. For the rook, these are (0, 1), (0, -1), (1, 0), and (-1, 0). For the bishop, they are (1, 1), (1, -1), (-1, 1), and (-1, -1).[56]
- **Iterative Path Checking:** The GetAvailableMoves method will loop through each of its direction vectors. For each direction, it will start a nested loop that moves one step at a time along that path.[57]
    - In each step, it checks the status of the new square.
    - If the square is empty, it is added to the list of valid moves, and the loop continues in that direction.
    - If the square contains an enemy piece, it is added as a valid capture move, and the loop for that direction *breaks*, as the path is now blocked.
    - If the square contains a friendly piece, the loop for that direction breaks immediately, as the piece cannot move to or through that square.

14.2 The Queen

The queen's power comes from its ability to move like both a rook and a bishop. This makes its implementation straightforward: its GetAvailableMoves method will simply combine the direction vectors of both the rook and the bishop and run the same iterative path-checking logic on all eight directions.[56]

Chapter 15: The King's Steps and Turn Management

The king's movement is simple, but its safety is the central concern of the game.

15.1 King Movement

Similar to the knight, the king's basic movement involves checking a set of adjacent squares. Its direction vectors are the same as the queen's, but it can only move one step in any direction. The GetAvailableMoves method will iterate through the eight

adjacent squares and add any that are within bounds and not occupied by a friendly piece.[58] The logic for preventing the king from moving into check will be handled later, during the legal move filtering stage.

15.2 Turn Management

After a player successfully completes a move, the turn must pass to the opponent. This is managed by the GameManager.cs script.

- A private PlayerColor currentPlayer; variable will track whose turn it is.
- A public void NextPlayer() method will be created. This method simply switches the value of currentPlayer:

```
C#
private void NextPlayer()
{
    currentPlayer = (currentPlayer == PlayerColor.White)? PlayerColor.Black :
PlayerColor.White;
}
```

This method will be called by the GameManager at the end of the MovePiece() function, ensuring a seamless transition between turns.[42]

Chapter 16: Special Move: Implementing Castling

Castling is a unique defensive move involving the king and a rook. Its implementation requires several strict conditions to be met.[59]

1. **No Prior Movement:** Both the king and the chosen rook must not have moved previously in the game. This is tracked using the bool hasMoved flag in the Piece class, which is set to true the first time a piece moves.
2. **Clear Path:** The squares between the king and the rook must be empty.
3. **King Not in Check:** The king cannot be in check at the start of the move.
4. **No Passage Through Check:** The king cannot pass through a square that is under attack by an enemy piece.
5. **No Ending in Check:** The king cannot end up on a square where it would be in check.

The king's GetAvailableMoves method will include logic to check for these conditions.

If all are met, a special "castling move" will be added to its list of available moves. When this move is executed, the GameManager will be responsible for moving both the king and the rook to their correct final positions in a single turn.[61]

Chapter 17: Special Move: The En Passant Capture

En passant ("in passing") is a special pawn capture that can only occur immediately after an opponent's pawn makes a two-square advance.

- **Tracking the Previous Move:** To implement this, the GameManager must store information about the last move made. A variable, for instance private Move lastMove;, will be updated after every turn.[63]
- **Pawn's Capture Logic:** The pawn's GetAvailableMoves method will be updated. It will check if the lastMove was a two-square advance by an adjacent enemy pawn. If so, it will add the empty square "behind" the enemy pawn as a valid capture target.[61]
- **Executing the Capture:** When an en passant move is executed, the GameManager must not only move the capturing pawn but also find and destroy the captured enemy pawn, which is located on an adjacent square, not the target square.

Chapter 18: Special Move: Pawn Promotion

When a pawn reaches the farthest rank from its starting position, it must be promoted to another piece (queen, rook, bishop, or knight) of the same color.[42]

1. **Detecting Promotion:** After a pawn move is executed, the GameManager will check if the pawn has landed on the 8th rank (for white) or the 1st rank (for black).
2. **Pausing the Game and Displaying UI:** If a promotion is detected, the game's normal flow is paused. A UI panel is displayed, presenting the player with buttons to choose which piece they want.[65]
3. **Executing the Promotion:** Once the player makes a selection, the GameManager will:
    - Destroy the pawn GameObject.

- Remove the pawn from the logical board array.
- Instantiate a new prefab for the chosen piece (e.g., a Queen) at the pawn's final position.
- Add the new piece to the logical board array.
- Hide the promotion UI panel and resume the game by calling NextPlayer().[66]

Part 6: The Endgame: UI and Winning Conditions

The final section of the textbook brings the project to completion by implementing the logic that determines the outcome of the game and creating the user interface (UI) to communicate this and other vital information to the player. This includes the complex but critical logic for detecting check, checkmate, and stalemate, as well as displaying captured pieces and a final game-over screen.

Chapter 19: Detecting Check and Checkmate

The ultimate goal in chess is to place the opponent's king in checkmate. The logic for detecting this state is the culmination of all the piece movement rules.

19.1 The IsInCheck() Method

A fundamental helper method is needed to determine if a king is currently under attack. This method, likely located in the Board or GameManager class, will function as follows [67]:

1. It takes a PlayerColor as an argument.
2. It finds the position of that player's king on the logical board array.
3. It then iterates through every single piece of the *opposing* color.
4. For each opposing piece, it generates a list of all squares that piece is currently attacking (its pseudo-legal moves).
5. If the king's position is found within any of these attack lists, the method returns true. Otherwise, after checking all opposing pieces, it returns false.

19.2 The "Virtual Move" and Legal Move Filtering

A move is only truly legal if it does not result in the player's own king being in check. This rule handles all cases of pins and prevents the king from moving into an attacked square. The most robust way to enforce this is by simulating each potential move.[69]

The process for generating a final list of legal moves for a player is:

1. Generate a list of all *pseudo-legal* moves for every one of the player's pieces (i.e., all moves the pieces could make if the king's safety were not a concern).
2. For each pseudo-legal move in this list:
   a. Create a temporary, virtual copy of the logical board state.
   b. Apply the move to this virtual board.
   c. Call the IsInCheck() method on the virtual board to see if the player's own king is now under attack.
   d. If the king is not in check, the move is truly legal and is added to the final list of legal moves. If the king is in check, the move is illegal and is discarded.68

This "virtual move" validation is the cornerstone of a correct chess engine. It elegantly handles all complex scenarios without needing to write specific, brittle logic for every possible type of pin or discovered attack.

19.3 Checkmate and Stalemate Logic

With the ability to generate a complete and accurate list of all legal moves, detecting checkmate and stalemate becomes straightforward. This check is performed at the very beginning of a player's turn:

1. Generate the complete list of all legal moves for the current player using the virtual move filtering process described above.
2. If this list of legal moves is empty, the game is over. The outcome is determined by one final check:
   a. Call IsInCheck() for the current player.
   b. If IsInCheck() returns true (the king is in check and has no legal moves), it is Checkmate. The opposing player wins.67

   c. If IsInCheck() returns false (the king is not in check but has no legal moves), it is Stalemate. The game is a draw.70

Chapter 20: Building the Game's User Interface

The UI is the primary channel for communicating game state information to the player. Unity's UI system is built around the Canvas.

1. **Create a Canvas:** In the Hierarchy, create a UI Canvas via GameObject > UI > Canvas. This object will be the parent for all UI elements.
2. **Configure the Canvas Scaler:** Select the Canvas GameObject. In the Inspector, find the Canvas Scaler component. Set its UI Scale Mode to **Scale With Screen Size**. This ensures that the UI elements will automatically resize to look correct on displays of different resolutions and aspect ratios, which is crucial for building a game that can be deployed on multiple platforms.[2]
3. **Core UI Elements:** The primary UI elements that will be used are:
    - **Image:** Used for backgrounds, panels, and displaying sprites of captured pieces.[72]
    - **Text - TextMeshPro:** The modern and powerful text solution in Unity, used for labels, scores, and game-over messages. It offers superior visual quality and control over standard UI Text.[2]
    - **Button:** Used for interactive elements like a "Restart Game" button.[72]
    - **Panel:** A basic Image component often used as a container to group other UI elements together.

Chapter 21: Displaying Captured Pieces and the Game Over Screen

This chapter implements the final UI features to create a complete game loop.

21.1 Displaying Captured Pieces

Providing a visual list of captured pieces is a standard feature in chess interfaces.

1. **Create Capture Zones:** On the main Canvas, create two empty GameObjects or Panels, one for each player, to act as containers for the captured pieces. Position them on either side of the screen.
2. **Use Layout Groups:** Add a Horizontal Layout Group component to each container. This component will automatically arrange any child UI elements neatly in a row, handling spacing and alignment.
3. **Capture Logic:** In the GameManager's CapturePieceAt() method, when a piece is captured, it is not just destroyed. A reference to it is stored in a list (List<Piece> capturedPieces).
4. **UI Update:** A new method, UpdateCapturedPiecesUI(), will be called after each

capture. This method will clear the current icons in the capture zones and then iterate through the capturedPieces lists for both players, instantiating a UI.Image prefab for each captured piece and placing it as a child of the correct container.[42] The
Horizontal Layout Group will handle the rest.

21.2 The Game Over Screen

When checkmate or stalemate is detected, the game should clearly communicate the result.

1. **Create the Panel:** Create a new Panel GameObject on the Canvas. Make it large enough to cover most of the screen and give it a semi-transparent dark background color. Disable this panel by default.[73]
2. **Add Content:** Add a Text - TextMeshPro element to the panel to display the result (e.g., "Checkmate! White Wins") and a Button with the text "Restart Game".
3. Triggering the Screen: When the GameManager detects checkmate or stalemate, it will:
   a. Set a boolean flag like isGameOver = true to stop further game logic.
   b. Update the text on the game-over panel with the correct message.
   c. Enable the game-over panel GameObject, making it visible.
4. **Restart Logic:** The "Restart Game" button's OnClick() event will be wired up in the Inspector to call a public method in the GameManager. This method will reload the entire scene, effectively restarting the game from the beginning [73]:
   C#
   ```
   using UnityEngine.SceneManagement;

   public void RestartGame()
   {
      SceneManager.LoadScene(SceneManager.GetActiveScene().name);
   }
   ```

Chapter 22: Final Polish and Next Steps

With a fully functional 3D chess game complete, this final chapter offers suggestions for further development and polish, turning the project from a functional application

into a more refined game experience.

- **Sound Effects:** Add AudioSource components and script calls to play sounds for piece selection, movement, captures, and check notifications.
- **Piece Animations:** Instead of having pieces teleport from one square to another, use Coroutines and Vector3.Lerp to animate them smoothly over a short duration.
- **AI Opponent:** The logical separation of the game model makes it possible to add an AI player. A beginner-friendly AI can be implemented using the **Minimax** algorithm, which evaluates possible moves to a certain depth to find the best option.[47]
- **Multiplayer:** With a solid single-player foundation, the project could be extended to include local hot-seat multiplayer or even networked multiplayer using Unity's networking solutions.[76]
- **FEN and PGN Support:** For advanced functionality, implement parsers for Forsyth-Edwards Notation (FEN) and Portable Game Notation (PGN). This would allow the game to load specific board positions or even load and replay entire recorded games.[78]

Works cited

1. Game Development Software: Create 2D & 3D Games - Unity, accessed July 25, 2025, https://unity.com/games
2. Working with UI in Unity, accessed July 25, 2025, https://learn.unity.com/tutorial/working-with-ui-in-unity
3. RTS Tutorial - Part 2 - Elgar's Code Musings, accessed July 25, 2025, http://www.stormtek.geek.nz/rts_tutorial/part2.php
4. Tutorial: Creating Interactive Objects in Unity for Micrsoft Mesh, accessed July 25, 2025, http://ilovemesh.com/tutorial-creating-interactive-objects-in-unity-for-micrsoft-mesh/
5. Unity Lighting Basics - Thoughtbot, accessed July 25, 2025, https://thoughtbot.com/blog/unity-lighting-basics
6. How do you set different textures on different sides of a 3D object? i'm really stumped by this, I got ProBuilder and I still don't know how to do it. : r/unity - Reddit, accessed July 25, 2025, https://www.reddit.com/r/unity/comments/1g0m9xf/how_do_you_set_different_textures_on_different/
7. Looking for a tutorial on making a game similar to chess. : r/Unity3D - Reddit, accessed July 25, 2025, https://www.reddit.com/r/Unity3D/comments/1dagkk/looking_for_a_tutorial_on_making_a_game_similar/
8. Create a Chessboard with 2D Arrays - These Hallways - WordPress.com, accessed July 25, 2025,

https://thesehallways.wordpress.com/2017/09/07/create-a-chessboard-with-2d-arrays/
9. Create a grid in Unity - Perfect for tactics or turn-based games! - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=kkAjpQAM-jE
10. How To Create An RTS Camera Controller - GameDev Academy, accessed July 25, 2025, https://gamedevacademy.org/unity-rts-camera-tutorial/
11. Scene view navigation - Unity - Manual, accessed July 25, 2025, https://docs.unity3d.com/Manual/SceneViewNavigation.html
12. What is the best way to make a top down camera move? : r/Unity3D - Reddit, accessed July 25, 2025, https://www.reddit.com/r/Unity3D/comments/10uqsrn/what_is_the_best_way_to_make_a_top_down_camera/
13. Unity Camera Controls, Part 1: Core Functionality - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=Jl2pB26ZOzI
14. How to make RTS Camera Movement in Unity - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=cfjLQrMGEb4
15. Showcase your work with lighting - Unity Learn, accessed July 25, 2025, https://learn.unity.com/pathway/creative-core/unit/lighting/tutorial/670feaaaedbc2a005441e630
16. Place Light components - Unity - Manual, accessed July 25, 2025, https://docs.unity3d.com/6000.1/Documentation/Manual/UsingLights.html
17. Dramatic Lighting of the Chess Board - Show - GameDev.tv, accessed July 25, 2025, https://community.gamedev.tv/t/dramatic-lighting-of-the-chess-board/169196
18. Light Up Your Game : Unity Lighting Essentials! - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=5rxMdiCkQGk
19. General best practices for fully enclosed indoor lighting? : r/Unity3D - Reddit, accessed July 25, 2025, https://www.reddit.com/r/Unity3D/comments/8fn3i0/general_best_practices_for_fully_enclosed_indoor/
20. Tips for lighting a 3D isometric game in Unity? : r/gamedev - Reddit, accessed July 25, 2025, https://www.reddit.com/r/gamedev/comments/1m14zay/tips_for_lighting_a_3d_isometric_game_in_unity/
21. LIGHTING in Unity - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=VnG2gOKV9dw
22. Free 3D Chess Models - Available For Download On TurboSquid, accessed July 25, 2025, https://www.turbosquid.com/Search/3D-Models/free/chess
23. Free 3D Chess Board Models - Available For Download On TurboSquid, accessed July 25, 2025, https://www.turbosquid.com/Search/3D-Models/free/chess-board
24. Free Chess 3D Models | CGTrader, accessed July 25, 2025, https://www.cgtrader.com/free-3d-models/chess
25. Free Chessboard 3D Models - CGTrader, accessed July 25, 2025, https://www.cgtrader.com/free-3d-models/chessboard
26. Chess 3D Models for Free - Download Free 3D - Clara.io, accessed July 25, 2025,

https://clara.io/library?query=chess&gameCheck=true
27. Chess Free 3D Models download - Free3D, accessed July 25, 2025, https://free3d.com/3d-models/chess
28. Chess Pieces & Board | 3D Props - Unity Asset Store, accessed July 25, 2025, https://assetstore.unity.com/packages/3d/props/chess-pieces-board-70092
29. Importing a model - Unity - Manual, accessed July 25, 2025, https://docs.unity3d.com/6000.1/Documentation/Manual/ImportingModelFiles.html
30. Importing 3D Models into Unity: A How-To - Coohom, accessed July 25, 2025, https://www.coohom.com/article/how-to-import-3d-models-into-unity
31. Meshy to Unity: Quick Guide to Import 3D Models & Animations - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=D0qUlOGyjgM
32. Importing Models - Unity User Manual 2021.3 (LTS), accessed July 25, 2025, https://docs.unity.cn/es/2021.1/Manual/ImportingModelFiles.html
33. Blender Tutorial: Creating a Chess Piece for 3D Printing | 3D ..., accessed July 25, 2025, https://i.materialise.com/blog/en/tutorial-creating-a-simple-chess-piece-in-blender/
34. Modelling a chess set (II) - Blender Artists Community, accessed July 25, 2025, https://blenderartists.org/t/modelling-a-chess-set-ii/1492534
35. Unity: How to Add a Texture to a Material - Wayline, accessed July 25, 2025, https://www.wayline.io/blog/unity-how-to-add-texture-to-material
36. How to Apply Material to 3D Model in Unity - Coohom, accessed July 25, 2025, https://www.coohom.com/article/how-to-apply-material-to-3d-model-in-unity
37. How to apply textures to 3d objects in Unity [English] - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=owy4VprOTFs
38. Combining meshes in Unity to a single mesh with unique materials | by Oleg Vashenkov | Medium, accessed July 25, 2025, https://medium.com/@oleg.vashenkov/combining-meshes-in-unity-to-a-single-mesh-with-unique-materials-81339356c281
39. Make a "board game" in Unity : r/Unity3D - Reddit, accessed July 25, 2025, https://www.reddit.com/r/Unity3D/comments/akmg7p/make_a_board_game_in_unity/
40. New to game dev especially on Unity trying to make a chess game with AI would appreciate some help . : r/Unity2D - Reddit, accessed July 25, 2025, https://www.reddit.com/r/Unity2D/comments/bo1eta/new_to_game_dev_especially_on_unity_trying_to/
41. Chess Game in Unity Tutorial! Part 1: Architecture and Board Generation - YouTube, accessed July 25, 2025, https://m.youtube.com/watch?v=cWgo0ak_8sE&pp=ygUNI2NoZXNzYnJvYWRjYQ%3D%3D
42. How to Make a Chess Game with Unity - Kodeco, accessed July 25, 2025, https://www.kodeco.com/5441-how-to-make-a-chess-game-with-unity/page/3
43. In Unity, how do I correctly implement the singleton pattern?, accessed July 25, 2025,

https://gamedev.stackexchange.com/questions/116009/in-unity-how-do-i-correctly-implement-the-singleton-pattern
44. Spawning Chess Pieces - 2/5 [Unity tutorial 2021][C#] - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=INcZNkU-XUw
45. How to Make a Chess Game with Unity - Kodeco, accessed July 25, 2025, https://www.kodeco.com/5441-how-to-make-a-chess-game-with-unity
46. Algorithm C# unity chess game - Stack Overflow, accessed July 25, 2025, https://stackoverflow.com/questions/39933973/algorithm-c-sharp-unity-chess-game
47. From Beginner to Checkmate: Building a Chess Engine | by ATHARVA MOTE | Medium, accessed July 25, 2025, https://medium.com/@atharva.mote/from-beginner-to-checkmate-building-a-chess-engine-8fa6fc34f517
48. C# - which data structure to use - manipulate and store chess position in 180 bits?, accessed July 25, 2025, https://stackoverflow.com/questions/12854409/c-sharp-which-data-structure-to-use-manipulate-and-store-chess-position-in-1
49. rudzen/ChessLib: C# chess library containing a complete data structure and move generation. - GitHub, accessed July 25, 2025, https://github.com/rudzen/ChessLib
50. Programming a Chess Game in C# | Part 3 - Pieces & The Board - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=NUNlVjt82m8
51. Raycast in Unity. Raycasting is a very common way to... | by Paul ..., accessed July 25, 2025, https://medium.com/@pkillman2000/raycast-in-unity-4c1895dc33d6
52. Detecting Objects with Raycasts in Unity3D - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=vqc9f7HU-Vc
53. Unity Tutorial: 3 Ways to Indicate Selected Units | Part 1 - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=OOkVADKo0IM
54. Outline Object on Pointer Hover and Selection at Runtime in Unity - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=qYnAkMGbgwo
55. Chess Game in Unity Tutorial! Part 2: Piece Movement and Input System. - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=8W-MJlgAz8U
56. Programming a Chess Game in C# | Part 5 - Generating Moves I - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=RDD48hlgAqU
57. Moving in a grid with certain number of steps - Unity GameDev - Stack Overflow, accessed July 25, 2025, https://stackoverflow.com/questions/65691956/moving-in-a-grid-with-certain-number-of-steps-unity-gamedev
58. Create an Online Chess Game - Regular Moves! - 4/5 [Unity tutorial 2021][C#] - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=lOEl1iNR3dc
59. Programming a Chess Game in C# | Part 13 - Castling - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=ECBgqxTkkgk
60. Chess TDD 48: Getting Started with Castling - DaedTech, accessed July 25, 2025,

https://daedtech.com/chess-tdd-48-getting-started-with-castling/
61. Chess Special Moves - Castling & Promotion - 2/3 [Unity tutorial 2021][C#] - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=syWLqjl1YQo
62. Chess Special Moves - En Passant - 1/3 [Unity tutorial 2021][C#] - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=2qua0_SO_M8
63. Need help implementing en passant Pawn capture and promotion - Stack Overflow, accessed July 25, 2025, https://stackoverflow.com/questions/24686803/need-help-implementing-en-passant-pawn-capture-and-promotion
64. Programming a Chess Game in C# | Part 14 - En Passant - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=X6ulMBsxte4
65. Programming a Chess Game in C# | Part 12 - Pawn Promotions - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=gSvPdoYmbXk
66. Chess Game in Unity Tutorial! Part 3: Finishing the Project. - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=9-6EcY1qLug
67. programming - Check for checkmate - Chess Stack Exchange, accessed July 25, 2025, https://chess.stackexchange.com/questions/13871/check-for-checkmate
68. Programming a Chess Game in C# | Part 8 - Detect Check & Legal Moves - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=eVL1dCioIVc
69. Chess | Unity C# Case Study by Alexander Larsen, accessed July 25, 2025, https://alexanderlarsen.com/case-studies/chess/
70. Check of checkmate in chess - algorithm - Stack Overflow, accessed July 25, 2025, https://stackoverflow.com/questions/30401046/check-of-checkmate-in-chess
71. Programming a Chess Game in C# | Part 9 - Checkmate & Stalemate - YouTube, accessed July 25, 2025, https://www.youtube.com/watch?v=oJpWFktFky8
72. UI Components - Unity Learn, accessed July 25, 2025, https://learn.unity.com/tutorial/ui-components
73. Lesson 5.3 - Game Over - Unity Learn, accessed July 25, 2025, https://learn.unity.com/pathway/junior-programmer/unit/user-interface/tutorial/lesson-5-3-game-over-2
74. Unity Game Over Screen - homeandlearn, accessed July 25, 2025, https://www.homeandlearn.co.uk/games-programming/game-over-screen.html
75. DarshanMaradiya/Unity-3D-AI-Chess: Minor Project: The ... - GitHub, accessed July 25, 2025, https://github.com/DarshanMaradiya/Unity-3D-AI-Chess
76. Complete Chess Game Tutorial - YouTube, accessed July 25, 2025, https://www.youtube.com/playlist?list=PLmcbjnHce7SeAUFouc3X9zqXxiPbCz8Zp
77. rlzicar/Chessticle - GitHub, accessed July 25, 2025, https://github.com/rlzicar/Chessticle
78. Chess Game Tutorial : r/unity_tutorials - Reddit, accessed July 25, 2025, https://www.reddit.com/r/unity_tutorials/comments/ii57f8/chess_game_tutorial/
79. Chess Program in C# - CodeProject, accessed July 25, 2025, https://www.codeproject.com/Articles/36112/Chess-Program-in-C

80. c# - Create FEN string in Unity after reading Chess Board - Stack Overflow, accessed July 25, 2025, https://stackoverflow.com/questions/54465630/create-fen-string-in-unity-after-reading-chess-board