Variables are declared to control the player's movement and behaviour. The difference between public and private variables is that public variables can be adjusted in the Unity Editor, while private variables are used internally to track the player's current state.

```
using UnityEngine;

/*
 * Developed player movement in order to move around
 *
 *
 * @author
 *  Andrei Costeniuc
 */

[RequireComponent(typeof(CharacterController))]
Unity-Skript (1 Objektverweis) | 0 Verweise
public class PlayerMovement : MonoBehaviour
{
    public Camera playerCamera;
    public float walkSpeed = 6f;
    public float runSpeed = 12f;
    public float lookSpeed = 2f;
    public float lookXLimit = 45f;
    public float defaultHeight = 2f;
    public float crouchHeight = 1f;
    public float crouchSpeed = 3f;

    private Vector3 moveDirection = Vector3.zero;
    private float rotationX = 0;
    private CharacterController characterController;

    private bool canMove = true;
```

Player's CharacterController is initialized, and the cursor is locked and made invisible. This is to control the player's movement with the mouse.

```
    Unity-Nachricht | 0 Verweise
    void Start()
    {
        characterController = GetComponent<CharacterController>();
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
```

Player's movement is calculated based on user input (keyboard and mouse in our case). Forward and right movement are transformed according to the player's look direction. If the player is running (by holding the left Shift key), a higher speed is used.

```
Unity-Nachricht | 0 Verweise
void Update()
{
    Vector3 forward = transform.TransformDirection(Vector3.forward);
    Vector3 right = transform.TransformDirection(Vector3.right);

    bool isRunning = Input.GetKey(KeyCode.LeftShift);
    float curSpeedX = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Vertical") : 0;
    float curSpeedY = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Horizontal") : 0;
    float movementDirectionY = moveDirection.y;
    moveDirection = (forward * curSpeedX) + (right * curSpeedY);
```

Code checks if the C key is pressed to make the player crouch. When the player crouches, the height of the CharacterController is reduced and the walk and run speed are set to the crouch speed. If the C key is not pressed, height and speed return to their default values.

```
    if (Input.GetKey(KeyCode.C) && canMove)
    {
        characterController.height = crouchHeight;
        walkSpeed = crouchSpeed;
        runSpeed = crouchSpeed;

    }
    else
    {
        characterController.height = defaultHeight;
        walkSpeed = 6f;
        runSpeed = 12f;
    }
```

Movement that has been calculated is applied to the CharacterController. The rotation of the player and the camera is calculated and applied based on the mouse movement. The vertical rotation of the camera is clamped to keep the player's viewing angle realistic.

```
        characterController.Move(moveDirection * Time.deltaTime);

        if (canMove)
        {
            rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;
            rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);
            playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);
            transform.rotation *= Quaternion.Euler(0, Input.GetAxis("Mouse X") * lookSpeed, 0);
        }
    }
}
```