Technical Description:

Progress Bar and KKProgressBar - This is used to show the player their attack cooldown, the right side of the screen shows an image and the scripts uses another image to go on top of the lower opacity image and make it seems like it is filling up to show the player that their basic attack is ready.

```
//If the current fill is less than the maximum value (The bar is not full) and the countdown is more than 0 (The move is recharging)
if (spell.timer > 0)
{
    //Percentage of the timer for the progress bar
    float percent = spell.timer / spell.cooldown;
    //Fills the progress bar starting from the bottom (Because Lerp is (1,0) if it was (0,1) then progress bar will start from the top and go down)
    progressbar.fillAmount = Mathf.Lerp(1, 0, percent);

}
else if (spell.timer <= 0)
{
    return;
}
```

 - Ready to Use  - Filling Up

Destroy Particles - This script is used specifically for the Spellcaster class so that when a user is using that class and they shoot the particle that is spawned will be destroyed after a certain amount of time and will check if the particle is still active in the scene.

```
public class DestroyParticles : MonoBehaviourPun
{
    private ParticleSystem ps;
    // Start is called before the first frame update
    void Start()
    {
        //Find the particle systems in the child of this game object
        ps = this.gameObject.transform.GetChild(0).GetComponent<ParticleSystem>();
    }

    // Update is called once per frame
    void Update()
    {
        //If the particle system is not alive then destroy the game Object
        if (!ps.IsAlive())
        {
            photonView.RPC( methodName: "DestroyObject", RpcTarget.All, this.GetComponent<PhotonView>().ViewID);
        }
    }

    [PunRPC]
    void DestroyObject(int go)
    {
        //Find the Id of the Game Object that needs to be destroyed
        Destroy(PhotonView.Find(go).gameObject);
    }
}
```

Goal - This script checks whether the treasure from the opposite team has been dropped off into the circle of the team the player is on. If they successfully drop the treasure chest into the circle they earn a point.

```
Unity Message | 0 references
private void OnTriggerEnter(Collider col)
{
    //If the Treasure is dropped into the Blue Goal
    if (this.gameObject.name == "Blue Goal Trigger" && col.gameObject.tag == "RedTreasure")
    {
        //Blue Team will gain a point
        photonView.RPC("IncreaseBlueScore", RpcTarget.All);
        Debug.Log(col.gameObject.GetComponent<PhotonView>().ViewID);
        //Destroy The Treasure Game Object For Everyone
        photonView.RPC("DestroyObject", RpcTarget.All, col.gameObject.GetComponent<PhotonView>().ViewID);

    }
    else if (this.gameObject.name == "Red Goal Trigger" && col.gameObject.tag == "BlueTreasure") // The treasure is dropped into the Red Goal
    {
        //red Team will gain a point
        photonView.RPC("IncreaseRedScore", RpcTarget.All);
        //Destroy The Treasure Game Object For Everyone
        Debug.Log(col.gameObject.GetComponent<PhotonView>().ViewID);
        photonView.RPC("DestroyObject", RpcTarget.All, col.gameObject.GetComponent<PhotonView>().ViewID);
    }
}
```

```
void IncreaseBlueScore()
{
    //Increases the score of the Blue Team
    nm.bluescore++;
}
```

Health - This script handles all the health-related functions regarding the player. There is a TakeDamage() function that handles the player taking damage if the related attack hits them and if they are hit then hit stun will be enabled for a brief amount of time. This health script also handles collision detection so that if the spellcaster shoots or the karate kid attacks then the appropriate function is called. If the player's health has reached 0 then they die and they will be destroyed. Then they will be able to pick another class and respawn back in the game after the respawn timer has reached 0. PhotonView holds a reference to the player's transform, scale and, rotation as well as the prefab so that it correctly applies the right damage and stun to the player as well as the respawning of the correct player that has died.

```
public class Health : MonoBehaviourPunCallbacks, IPunObservable
{
    public float health = 100;

    Spellcaster spell;

    KarateKid kid;

    NetworkManager nm;

    PlayerController pc;

    Teams myClass;

    public float deathtimer, hitstuntimer, hitstun;
    public bool dead = false;
    public bool hitStun = false;

    [SerializeField] Animator _playeranim;

    // Start is called before the first frame update
    void Start()
    {
        myClass = GetComponent<Teams>();
        if (myClass.classid == Teams.chosenClass._Spellcaster)
        {
            spell = GetComponent<Spellcaster>();

        }
        else if (myClass.classid == Teams.chosenClass._Karate)
        {
            kid = GetComponent<KarateKid>();
        }
        nm = GameObject.Find("NetworkManager").GetComponent<NetworkManager>();
        //Get The Player Controller from this Player Object
        pc = GetComponent<PlayerController>();
        //Grab Player Animator
        _playeranim = GetComponent<Animator>();

        deathtimer = 5.0f;
    }
```

```
void Update()
{
    // When the health is less than or equal to 0 then destroy the Player Game Object
    if (health <= 0)
    {
        //Start the countdown
        deathtimer -= Time.deltaTime;
        //Disable Scripts
        pc.enabled = false;
        //Makes it so Players cant attack no more
        ClassSwitch( onoff: false);


        if (dead == false)
        {
            _playeranim.SetTrigger( name: "Death");
            dead = true;
        }

        if (deathtimer == 0 || deathtimer <= 0)
        {
            //Tells the Network Manager that the player is not alive which means display the Respawn Button
            nm.isAlive = false;
            //Destroy The Player Game Object
            photonView.RPC( methodName: "DestroyObject", RpcTarget.All, GetComponent<PhotonView>().ViewID);
        }
    }
    else
    {
        //Player cant move until the timer is done
        if (hitstuntimer > 0 && health > 0)
        {
            hitstuntimer -= Time.deltaTime;

            if (hitStun == false)
            {
                    _playeranim.SetTrigger( name: "hitStun");
                    hitStun = true;
            }

        }
        else
        {
            //Re enable the Player Controller
            pc.enabled = true;
            //Makes it so Players can attack after being hit
            ClassSwitch( onoff: true);
        }
    }
}
```

```
void TakeDamage(float damagetaken)
{
    //Makes sure that when you deal damage you dont take damage from your own attack
    health -= damagetaken;
    Debug.Log( message: "Lost Health");
    //Make it so player cant move
    pc.enabled = false;
    //Makes it so Players cant attack no more
    ClassSwitch( onoff: false);
    //Add Hitstun
    hitstuntimer = hitstun;
    hitStun = false;
   /* if (health > 0)
    {
        _playeranim.SetTrigger("hitStun");
    }*/

    if (pc.carrying) //If Player is carrying
    {
        //Make Player drop the Treasure
        pc.SetCarrying( isPickingUp: false);
        //Find The Treasure Trigger Script in all the child of this game Object and call the Drop Treasure function
        GetComponentInChildren<TreasureTrigger>().DropTreasure(pc);
    }



}
```

```
private void OnParticleCollision(GameObject col)
{
    //Makes sure that when you get hit its from someone else and not yourself
    if (photonView.IsMine)
    {
        //Gets The "OwnerOfSpell" script from the spell the player collided with (We get the parent object since the particle sets off the collision)
        OwnerOfSpell ownerofspell = col.transform.parent.gameObject.GetComponent<OwnerOfSpell>();
        Debug.Log( message: "Ok: " + ownerofspell);

        //Get the Owner Id Of The Spell
        int ownerid = ownerofspell.GetOwner();

        //Find the player id of the owner of the attack and get the Spellcaster script
        spell = PhotonView.Find(ownerid).GetComponent<Spellcaster>();
        Teams ct = PhotonView.Find(ownerid).GetComponent<Teams>();

        Debug.Log( message: "Owner Of Attack: " + PhotonView.Get(PhotonView.Find(ownerid).gameObject));

        //Make Sure the player isnt get hit by its own spell
        if (ownerid != this.GetComponent<PhotonView>().ViewID) //If they are not getting hit by their own spell (So being attacked)
        {
            Debug.Log( message: "Being attacked");

            //Depending on the attack the player will lose a certain amount of health
            if (col.gameObject.tag == "Basic Attack")
            {
                if(ct.teamid != this.GetComponent<Teams>().teamid)
                {
                    //Attacked Player will take damage and check how much damage the attack should deal from the owner of the attack
                    TakeDamage( damagetaken: spell.DealDamage());
                    //Destroy The Spell Game Object For Everyone
                    photonView.RPC( methodName: "DestroyObject", RpcTarget.All, col.transform.parent.gameObject.GetComponent<PhotonView>().ViewID);
                    Debug.Log( message: "Basic Attack has hit " + col.gameObject.name);
                } else
                {
                    Debug.Log( message: "Friendly Fire on the red team");
                }



            }
        }
```

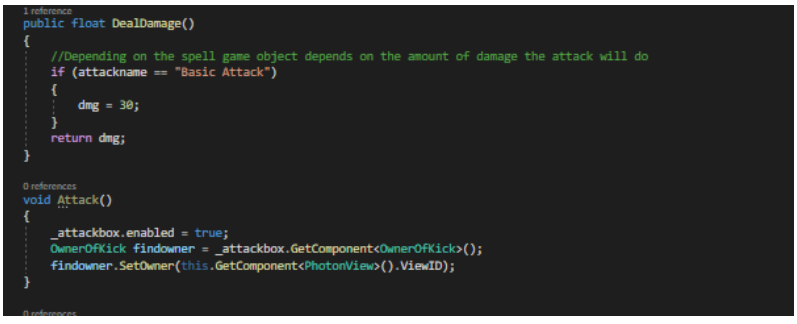
```
        else //If they are hitting themselves
        {
            Debug.Log( message: "Hitting yourself");
            //Destroy The Spell Game Object For Everyone
            photonView.RPC( methodName: "DestroyObject", RpcTarget.All, col.transform.parent.gameObject.GetComponent<PhotonView>().ViewID);
            //Allow the player to shoot again without any cooldown
            spell.timer = 0;
        }

    }

}
```

```
private void OnTriggerEnter(Collider collision)
{
    //Makes sure that when you get hit its from someone else and not yourself
    if (photonView.IsMine)
    {
        if (collision.tag == "Basic Attack2") //Makes sure its Colliding with an Attack
        {
            //Gets The "OwnerOfKick" script from the spell the player collided with
            OwnerOfKick ownerofkick = collision.gameObject.GetComponent<OwnerOfKick>();
            Debug.Log( message: "Victim = " + this.gameObject.name + "Hit by: " + collision.gameObject.name);
            Debug.Log( message: "Ok: " + ownerofkick);


            //Get the Owner Id Of The Kick
            int ownerid = ownerofkick.GetOwner();

            //Find the player id of the owner of the attack and get the Spellcaster script
            kid = PhotonView.Find(ownerid).GetComponent<KarateKid>();
            Teams ct = PhotonView.Find(ownerid).GetComponent<Teams>();

            Debug.Log( message: "Owner Of Attack: " + PhotonView.Get(PhotonView.Find(ownerid).gameObject));

            //Make Sure the player isnt get hit by its own spell
            if (ownerid != this.GetComponent<PhotonView>().ViewID) //If they are not getting hit by their own spell (So being attacked)
            {
                Debug.Log( message: "Being attacked");

                //Depending on the attack the player will lose a certain amount of health
                if (collision.gameObject.tag == "Basic Attack2")
                {
                    if (ct.teamid != this.GetComponent<Teams>().teamid)
                    {
                        //Attacked Player will take damage and check how much damage the attack should deal from the owner of the attack
                        TakeDamage( damagetaken: kid.DealDamage());
                        Debug.Log( message: "Basic Attack has hit " + collision.gameObject.name);
                    }
                    else
                    {
                        Debug.Log( message: "Friendly Fire on the red team");
                    }

                }
            }
        }
```

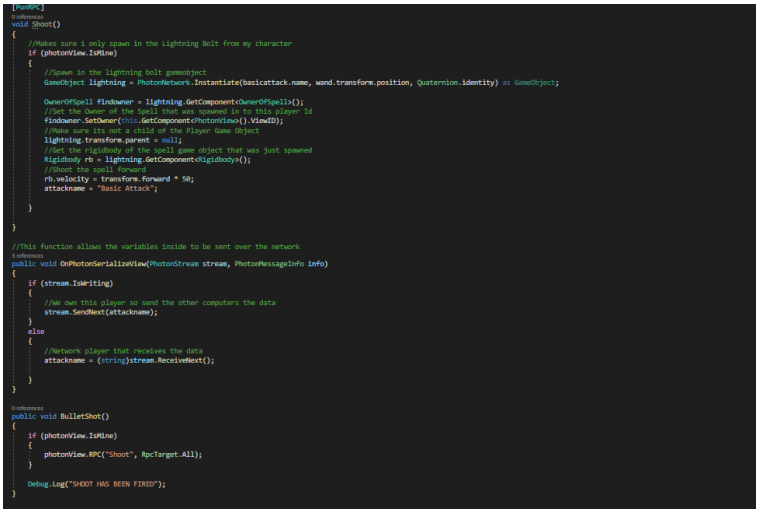
```
    //This function allows the variables inside to be sent over the network
    public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
    {
        if (stream.IsWriting)
        {
            //We own this player so send the other computers the data
            stream.SendNext(health);
        }
        else
        {
            //Network player that receives the data
            health = (float)stream.ReceiveNext();
        }
    }


    [PunRPC]
    void DestroyObject(int go)
    {
        //Find the Id of the Game Object that needs to be destroyed
        Destroy(PhotonView.Find(go).gameObject);
    }


    //Makes it so that we can turn the Class script on and off (Makes it whether or not the player can attack or not
    void ClassSwitch(bool onoff)
    {
        if (myClass.classid == Teams.chosenClass._Spellcaster)
        {
            spell.enabled = onoff;

        }
        else if (myClass.classid == Teams.chosenClass._Karate)
        {
            kid.enabled = onoff;
        }
    }

}
```

SpellCaster and KarateKid - These scripts handle the way the classes function. They both have their own ways of working but use the same assets and script code to function similar things such as attacking. However they both do different functions for their attacks, it is still called in similar manners.

Karate -



```
1 reference
public float DealDamage()
{
    //Depending on the spell game object depends on the amount of damage the attack will do
    if (attackname == "Basic Attack")
    {
        dmg = 30;
    }
    return dmg;
}

0 references
void Attack()
{
    _attackbox.enabled = true;
    OwnerOfKick findowner = _attackbox.GetComponent<OwnerOfKick>();
    findowner.SetOwner(this.GetComponent<PhotonView>().ViewID);
}
```

SpellCaster - 

```
[PunRPC]
0 references
void Shoot()
{
    //Makes sure i only spawn in the Lightning Bolt from my character
    if (photonView.IsMine)
    {
        //Spawn in the lightning bolt gameobject
        GameObject lightning = PhotonNetwork.Instantiate(basicattack.name, wand.transform.position, Quaternion.identity) as GameObject;

        OwnerOfSpell findowner = lightning.GetComponent<OwnerOfSpell>();
        //Set the Owner of the Spell that was spawned in to this player Id
        findowner.SetOwner(this.GetComponent<PhotonView>().ViewID);
        //Make sure its not a child of the Player Game Object
        lightning.transform.parent = null;
        //Get the rigidbody of the spell game object that was just spawned
        Rigidbody rb = lightning.GetComponent<Rigidbody>();
        //Shoot the spell forward
        rb.velocity = transform.forward * 50;
        attackname = "Basic Attack";

    }

}

//This function allows the variables inside to be sent over the network
3 references
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        //We own this player so send the other computers the data
        stream.SendNext(attackname);
    }
    else
    {
        //Network player that receives the data
        attackname = (string)stream.ReceiveNext();

    }
}

0 references
public void BulletShot()
{
    if (photonView.IsMine)
    {
        photonView.RPC("Shoot", RpcTarget.All);
    }

    Debug.Log("SHOOT HAS BEEN FIRED");
}
```

MainMenuManager - The Main Menu Manager allows the players to create and join the lobby, there is a timer that will go down and when it hits 0 the game will start for all players in the lobby no matter how many players are in the game, the timer will be decreased to 5 seconds if the lobby is full. The timer will be sent over the network to everyone in the lobby thanks to the OnPhotonSerializeView() Function.

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    //If we are in a room
    if (PhotonNetwork.InRoom)
    {
        players.text = "Players: " + PhotonNetwork.CurrentRoom.PlayerCount + " of " + PhotonNetwork.CurrentRoom.MaxPlayers;

        //Lists all the players in the room
        players.text += ":\n";
        Dictionary<int, Player> mydict = PhotonNetwork.CurrentRoom.Players;
        int i = 1;
        //Checks the names of each player
        foreach (var item in mydict)
            players.text += string.Format("{0,2}. {1}\n", (i++), item.Value.NickName);

        DisplayTime(timer);

        //If the timer variable is more than 0
        if (timer > 0)
        {
            // Then Timer will start counting down
            timer -= Time.deltaTime;
        }
        else
        {
            //When the timer reaches 0 the game will start
            StartGame();
        }

    }
```

```
3 references
public override void OnConnectedToMaster()
{
    //Once connected to Photon server...
    Debug.Log("OnConnectedToMaster was called by PUN.");
    //Allows us to create and join rooms
    PhotonNetwork.JoinLobby();

    //Gets the Players name and sets the text to the player name
    //playerName.text = PlayerPrefs.GetString("PlayerName");
}

3 references
public override void OnJoinedLobby()
{
    //When we have joined a lobby


}

//When you press the JoinRoom button
0 references
public void JoinRoom()
{

    //Sets the player name (Player name is set to PlayerPrefs)
    PlayerPrefs.SetString("PlayerName", playerName.text);
    PhotonNetwork.NickName = playerName.text;
    //Joins a random room
    PhotonNetwork.JoinRandomRoom();
}

3 references
public override void OnJoinedRoom()
{
    //Disable Join Button
    joinButton.gameObject.SetActive(false);
    //Make it so you can no longer input a name
    playerName.gameObject.SetActive(false);
    //Show the list of Players in the room
    players.gameObject.SetActive(true);
    //Show the Timer
    timertxt.gameObject.SetActive(true);
}
```

```
//This function allows the variables inside to be sent over the network
3 references
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        //We own this player so send the other computers the data
        stream.SendNext(timer);

    }
    else
    {
        //Network player that receives the data
        timer = (float)stream.ReceiveNext();

    }
}
```

NetworkManager - The Network Manager will spawn the Players into the Online lobby, the character they will be controlling depends on the team they decide to join and the class they pick. The network manager will also keep count of the score and it will keep checking if there is a winner. When there is a winner the game will pause with the team name that won and after a while, all the players will be kicked off the lobby and back into the main menu. The Score System can be seen by everyone in the lobby thanks to the OnPhotonSerializeView() function along with the win variable.

```
//Checks to see who the winner is
1 reference
void CheckScore()
{
    if (bluescore == 3)
    {
        win = true;
        bluewinnertext.gameObject.SetActive(true);
        bluewinnertext.text = "BLUE TEAM WINS";
        buttonLeave.gameObject.SetActive(false);
    }
    else if (redscore == 3)
    {
        win = true;
        redwinnertext.gameObject.SetActive(true);
        redwinnertext.text = "RED TEAM WINS";
        buttonLeave.gameObject.SetActive(false);
    }
```

```
0 references
public void blue_Team_Pick()
{
    //Dont do anything if the team is full
    if (bluePlayerCount == maxInTeam)
        return;
    //Choose Blue Team
    teamPick = 1;
    //Increase the amount of players in the team
    photonView.RPC("IncreaseBluePlayerCount", RpcTarget.All);
    //Default Pick The Spell Caster Character
    player = bp_SC_Prefab;
    //Debug.Log("I am blue team");
    //Blue Team Active Buttons
    blue_SpellClass.gameObject.SetActive(true);
    blue_WarriorClass.gameObject.SetActive(true);
    red_SpellClass.gameObject.SetActive(false);
    red_WarriorClass.gameObject.SetActive(false);
    //Disable Team Picking
    redTeam.gameObject.SetActive(false);
    blueTeam.gameObject.SetActive(false);
    //Spawn Location For Blue Team
    spawnLocation = new Vector3(33.0f, 2.7f, 30.0f);
    //Spawn Rotation For Blue Team
    spawnRotation = Quaternion.Euler(0, -90, 0);
    //Show The Score Of The Game
    bluescoretext.gameObject.SetActive(true);
    redscoretext.gameObject.SetActive(true);

}
```

```
//This function allows the variables inside to be sent over the network
3 references
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        //We own this player so send the other computers the data
        stream.SendNext(bluescore);
        stream.SendNext(redscore);
        stream.SendNext(win);
    }
    else
    {
        //Network player that receives the data
        bluescore = (int)stream.ReceiveNext();
        redscore = (int)stream.ReceiveNext();
        win = (bool)stream.ReceiveNext();
    }
}
```

OwnerOfKick and OwnerOfSpell - These 2 scripts are made to make sure that both classes' basic attacks can only be used by the person who spawns the prefab that they have control over. This is to make sure that not everyone's moves are activated when one person presses the button on command. It also makes sure the other player gets the data from the player shooting or doing an attack. These scripts are then referenced to the moves within the spellcaster and karate scripts.

```
public int owner;


//This function allows the variables inside to be sent over the network
3 references
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        //We own this player so send the other computers the data
        stream.SendNext(owner);
    }
    else
    {
        //Network player that receives the data
        owner = (int)stream.ReceiveNext();

    }
}
//Sets the owner of the Spell
1 reference
public void SetOwner(int ownerOfSpell)
{
    owner = ownerOfSpell;
}

//When called Returns the Owner Of The Spell
1 reference
public int GetOwner()
{
    return owner;
}
```

PlayerController - This script handles all the player movement variables and functions. It also allows the player to move using a rigidbody rather than translating it and this is done by adding the correct amount of force depending on what key they are pressing. It is also being referenced by PhotonView since each player should be able to control their own character that they are using. The script also controls the rigidbody drag so they move as if they are not floating character. It also controls the speed of the player if they are holding the treasure chest.

```
[RequireComponent(typeof(Animator))]
public class PlayerController : MonoBehaviourPun
{
    // Link: https://canvas.kingston.ac.uk/courses/19811/pages/pun-guided-programming-part-3-multiplayer-movement
    //public float turnSpeed = 180;
    //public float tiltSpeed = 180;
    //public float walkSpeed = 1;

    [Header("PlayerMovement")]
    public float _moveSpeed = 7f; //Player movement speed;
    public float _moveSlowSpeed = 3f; //Player movement speed;
    [SerializeField] private float _horizontalMovement;
    [SerializeField] private float _verticalMovement;
    [SerializeField] private float _movementMultiplier = 10f;
    [SerializeField] private Vector3 _moveDir; //Player move direction
    [SerializeField] private Rigidbody _playerRB; //Player Rigidbody
    public Animator _playeranim; // Player Animation Referrence

    [Header("GroundSettings")]
    [SerializeField] private float _playerHeight = 2f;
    [SerializeField] private bool _isGrounded;
    [SerializeField] private float _groundDrag = 6f; //Player ground Drag

    [Header("OtherSettings")]
    [SerializeField]
    public Transform fpcam;    // first person camera
    [SerializeField]
    TextMesh nickname;


    public bool carrying; // Player is carrying the Treasure, also used to notify the treasure that it is being carried
```

```
// Start is called before the first frame update
void Start()
{

    //photonView.IsMine - It only gets your client and only i can control it
    //If your player is there and your fp camera is there
    if (photonView.IsMine && fpcam != null)
    {
        //Gets player Rigidbody
        _playerRB = GetComponent<Rigidbody>();
        fpcam.GetComponent<Camera>().enabled = true;
        nickname.text = " ";

        //Grab Player Animator
        _playeranim = GetComponent<Animator>();

        carrying = false;
    }

    else  //If its not my client and its another player
    {
        //Gets the other players nickname
        nickname.text = photonView.Owner.NickName;
    }
}
```

```
// Update is called once per frame
void Update()
{

    //Makes sure i am controlling my own player
    if (photonView.IsMine)
    {
        _isGrounded = Physics.Raycast( origin: transform.position,  direction: Vector3.down,  maxDistance: _playerHeight / 2 + 0.1f);

        PlayerInput();
        ControlDrag();
        //Player Animation Parameter
        _playeranim.SetFloat( name: "Speed",  value: Mathf.Abs(_moveDir.x));
    }

    if (Camera.current != null)
    {
        //nicknames of other players are always facing towards me
        nickname.transform.LookAt(Camera.current.transform);
        nickname.transform.Rotate( xAngle: 0,  yAngle: 180,  zAngle: 0);
    }



}
```

```
    private void FixedUpdate()
    {
        MovePlayer();
    }

    //Player Movement Input
    void PlayerInput()
    {
        _horizontalMovement = Input.GetAxisRaw("Horizontal");
        _verticalMovement = Input.GetAxisRaw("Vertical");

        _moveDir = transform.forward * _verticalMovement + transform.right * _horizontalMovement;
    }

    //Move the player using force
    void MovePlayer()
    {
        if (_isGrounded)
        {
            _playerRB.AddForce(_movementMultiplier * _moveSpeed * _moveDir.normalized, ForceMode.Acceleration);
        }
    }

    //Adding drag to the player to not make them move as if they are floating
    void ControlDrag()
    {
        if (_isGrounded)
        {
            _playerRB.drag = _groundDrag;
        }
    }

    //So the Player knows when its carrying something or not
    public void SetCarrying(bool isPickingUp)
    {
        carrying = isPickingUp;
    }
}
```

PlayerLook - This script allows the player to control their camera, they are only able to control their camera if they are alive.

```
void Update()
{
    if (Input.GetKey(KeyCode.Escape))
    {
        Cursor.visible = true;
    }


    if (photonView.IsMine)
    {
        if (health.health > 0)
        {
            CameraInput();

            _cam.transform.localRotation = Quaternion.Euler(_xRot, 0, 0);
            transform.rotation = Quaternion.Euler(0, _yRot, 0);
        }
    }
}

1 reference
void CameraInput()
{
    _mouseX = Input.GetAxisRaw("Mouse X");
    _mouseY = Input.GetAxisRaw("Mouse Y");

    _yRot += _mouseX * _sensX * _multiplier;
    _xRot -= _mouseY * _sensY * _multiplier;

    _xRot = Mathf.Clamp(_xRot, 15f, 15f);
}
```

Teams - This simple script is used to set the teams for the prefabs spawned and is used in other scripts so that teammates can not hit one another.

```
Unity Script (9 asset references) | 12 references
public class Teams : MonoBehaviour
{
    //Gets the team ID for each prefab
    1 reference
    public enum chosenTeam { Red, Blue };
    public chosenTeam teamid;

    5 references
    public enum chosenClass { _Spellcaster, _Karate };
    public chosenClass classid;


}
```

Treasure Trigger - This script allows the player to hold a treasure chest when they are near it. Photon will know who has picked up the treasure and attach it to them. They are able to pick it up, put it down, and throw the treasure far to try and pass it to a teammate.

```
if (inTrigger) //If there is a player in the Trigger
{
    if (playerpv.IsMine) //Check to see if it is the Payer pressing the button
    {
        PlayerController pc = playerpv.GetComponent<PlayerController>();
        if (Input.GetKeyDown(KeyCode.F) && !isPickedUp && inTrigger && !pc.carrying) //If the F Key Is Pressed and the Treasure hasnt been picked up yet and there is still a player in the trigger and the player is not carrying
        {
            // The Player will pick up the Treasure
            photonView.RPC("AttachToPlayer", RpcTarget.All, playerid);
            Debug.Log("Picked Up Treasure");
            //Treasure Animation is true
            pc._playeranim.SetBool("Carrying", true);
            //Set Speed to Slow
            pc._moveSpeed = pc._moveSlowSpeed;
            //Notify the Player object that it is carrying something
            pc.SetCarrying(true);

        }
        else if (Input.GetKeyDown(KeyCode.G) && isPickedUp && pc.carrying) //If the G Key Is Pressed and the Treasure has been picked up
        {
            //Player Drops the Treasure
            photonView.RPC("DetachFromPlayer", RpcTarget.All);
            //Treasure will know it has been dropped
            isPickedUp = false;
            Debug.Log("Dropped Treasure");
            pc._playeranim.SetBool("Carrying", false);
            pc._moveSpeed = 7f;
            //Notify the Player object that it is carrying something
            pc.SetCarrying(false);
        }
        else if (Input.GetKeyDown(KeyCode.Space) && isPickedUp && canBeThrown && pc.carrying) //If the Space Key is pressed and the Treasure has been picked up and the Treasure can be thrown
        {
            photonView.RPC("ThrownFromPlayer", RpcTarget.All, playerid);
            pc._playeranim.SetBool("Carrying", false);
            pc._moveSpeed = 7f;
            //Notify the Player object that it is not carrying no more
            pc.SetCarrying(false);
        }
```

```
    [PunRPC]
    void ThrownFromPlayer(int IdOfPlayer)
    {
        PlayerController playerController = PhotonView.Find(IdOfPlayer).GetComponent<PlayerController>();
        DetachFromPlayer();
        rb.AddForce(PhotonView.Find(IdOfPlayer).transform.forward * thrownForce, ForceMode.Impulse);
        rb.AddForce(PhotonView.Find(IdOfPlayer).transform.up * thrownForce, ForceMode.Impulse);
        canBeThrown = false;
    }
```

```
    [PunRPC]
    void DetachFromPlayer()
    {
        //Becomes its own object again
        parentObject.gameObject.transform.parent = null;
        isPickedUp = false;
        //Put Gravity back on so that the Treasure falls down
        rb.useGravity = true;
        //Allows the Gravity to work for the treasure
        rb.isKinematic = false;
        //Set Collision to true
        parentObject.gameObject.GetComponent<BoxCollider>().enabled = true;
        Debug.Log("Dropped");
    }

    [PunRPC]
    void ThrownFromPlayer(int IdOfPlayer)
    {
        PlayerController playerController = PhotonView.Find(IdOfPlayer).GetComponent<PlayerController>();
        DetachFromPlayer();
        rb.AddForce(PhotonView.Find(IdOfPlayer).transform.forward * thrownForce, ForceMode.Impulse);
        rb.AddForce(PhotonView.Find(IdOfPlayer).transform.up * thrownForce, ForceMode.Impulse);
        canBeThrown = false;
    }

    public void DropTreasure(PlayerController pController)
    {
        //Player Drops the Treasure
        photonView.RPC("DetachFromPlayer", RpcTarget.All);
        //Treasure will know it has been dropped
        isPickedUp = false;
        Debug.Log("Dropped Treasure");
        pController._playeranim.SetBool("Carrying", false);
        pController._moveSpeed = 7f;
    }
}
```