Functionality:
First, we have implemented multiple game states. When the game is first started, the user is given a selection between single-player mode and multiplayer mode. In single-player mode, the player is loaded into the game and tries to get 10 power ups without falling off the map. If they lose, they get a lose screen with the reason they lost. If they win, they are congratulated with a win screen that says why they won. Both scenes will prompt the player to restart the game again or to quit to the main screen. In multiplayer mode, the player is loaded into a second GUI menu where they are given the selection to host or join a game.

This second GUI menu in multiplayer mode is our lobby system. One player will add their username and host the game to start themself as the server. The second player will add their username and then join the game to add themself as a client to the server. Once both players have joined, the players can start the game. Either player can then click the start button to do this. Both players are then transferred into the game scene when the players can then race to get the 10 power ups first. Their usernames are displayed above them.

We have set up smooth handling of both connections and disconnections. If a player joins midway through a game (started game), the player is added as a client to the existing server and both players are updated with each other. Additionally, if either player disconnects (server or client) then they are removed from the game and the other player wins the game by default (will be given a winning screen).
We have set up game end conditions for both modes as well. When the single player ends, they have the choice to restart the game or to return to the main screen. When the multiplayer ends, they have the same options as they do in singleplayer (restart or exit), but here the restart takes you back to the lobby instead of restarting the game automatically. In the lobby, players can host, connect to, and start a new game.

For our extension, we have set up dead reckoning. This allows the game system to predict the player's position based on current movement so that the game doesn't have to send as many movement update packets to the other player. This allows for smoother gameplay, as it reduces latency by not sending and processing as much information.

Code Layout:
We have a few GDScripts that handle a majority of the GUI and the networking. We first have a main menu script that handles single player vs multiplayer. We then have a separate start menu for setting up networking within the multiplayer mode. That file takes care of a player connecting as host, a player connecting as a client, and taking a username as an input. It also handles starting the actual game. Lastly, we have a few networking components in the field script that simply allow the networked players to be all added and set up within the actual game. In the player scene, we have a multiplayer synchronizer that tracks each player's movement, rotation, and the amount of lives. Lastly, we have a bit of networking being handled within the player's script where we take care of sending updated user data as well as dead reckoning for improved networking.

Code Segments:

Options to pick singleplayer or multiplayer where we set up and change scene accordingly

```
func _on_singleplayer_pressed():
    GameManager.Players[1] = {
    "name": "default",
    "id" : 1,
    "score" : 0
    }
    GameManager.mode = "singleplayer"
    get_tree().change_scene_to_file("res://scenes/main.tscn")


func _on_multiplayer_pressed():
    GameManager.mode = "multiplayer"
    get_tree().change_scene_to_file("res://scenes/start_menu.tscn")
```

Main function that handles setting up a server when the host option is selected within the lobby system.

```
func _on_host_button_down():
    if not GameManager.readied:
        peer = null
        multiplayer.peer_connected.disconnect(peer_connected)
        multiplayer.peer_disconnected.disconnect(peer_disconnected)
        multiplayer.connected_to_server.disconnect(connected_to_server)
        multiplayer.connection_failed.disconnect(connection_failed)
        initialize_game()

    peer = ENetMultiplayerPeer.new()
    var error
    if GameManager.readied :
        error = peer.create_server(port, 2) # 2 player game
    else :
        error = peer.create_server(port + 1, 2) # use new port
    if error != OK:
        print("cannot host: " + error_string(error))
        return
    peer.get_host().compress(ENetConnection.COMPRESS_RANGE_CODER)

    multiplayer.set_multiplayer_peer(peer)
    print("Waiting for players!")
    SendPlayerInformation($UsernameInput.text, multiplayer.get_unique_id())

    start_button.disabled = false
```

Main join function that handles setting up a client when the join option is selected within the lobby system.

```
func _on_join_button_down():
    peer = ENetMultiplayerPeer.new()
    peer.create_client(Address, port)
    peer.get_host().compress(ENetConnection.COMPRESS_RANGE_CODER)
    multiplayer.set_multiplayer_peer(peer)

    start_button.disabled = false
```

Main RPC call that adds player information and sends all the players' information through the server.

```
@rpc("any_peer")
func SendPlayerInformation(name, id):
    if !GameManager.Players.has(id):
        GameManager.Players[id] = {
            "name": name,
            "id" : id,
            "score" : 0
        }

    if multiplayer.is_server():
        for i in GameManager.Players:
            SendPlayerInformation.rpc(GameManager.Players[i].name, i)
```

Main RPC call that starts the game for all connected players.

```
@rpc("any_peer", "call_local")
func start_game():
    if get_node_or_null("../main") :
        GameManager.new_client_connected = true
    else :
        var scene = load("res://scenes/main.tscn").instantiate()
        get_tree().root.add_child(scene)
        self.hide()
```

RPC calls that both send updated food position to the server and get broadcasted from the server to all clients. Updates remote changes to food positions for server and all clients locally.

```
void Player::move_food(String food, Vector3 pos) {
    get_node<Food>("../Food" + food)->set_position(pos);
    rpc("broadcast_food", food, pos);
}

void Player::broadcast_food(String food, Vector3 pos) {
    get_node<Food>("../Food" + food)->set_position(pos);
}
```

Code that sets up and spawns the players in the game field.

```gdscript
# Called when the node enters the scene tree for the first time.
func _ready():
	var index = 0
	for i in GameManager.Players:
		var currentPlayer = PlayerScene.instantiate()
		currentPlayer.set_multiplayer_authority(GameManager.Players[i].id)
		if (i == 1):
			Player1 = currentPlayer
			Player1.get_node("Label3D").text = GameManager.Players[i].name#Player1.get_name()
		else:
			Player2 = currentPlayer
			Player2.get_node("Label3D").text = GameManager.Players[i].name#Player2.get_name()
		currentPlayer.name = str(GameManager.Players[i].id) # can access in c++ with get_name()
		add_child(currentPlayer)
		for spawn in get_tree().get_nodes_in_group("SpawnPoint"):
			if spawn.name == "Spawn" + str(index):
				currentPlayer.global_position = spawn.global_position
		index += 1

	if GameManager.Players.size() == 2 :
		for j in GameManager.Players:
			if (j == 1):
				Player2.set_other_id(GameManager.Players[j].id)
			else:
				Player1.set_other_id(GameManager.Players[j].id)
```

Allows newly joined clients to a started game to be updated on the host machine.

```gdscript
func _process(delta):
	if GameManager.new_client_connected and multiplayer.get_unique_id() == 1 :
		var index = 0
		for i in GameManager.Players:
			if i != 1 :
				var currentPlayer = PlayerScene.instantiate()
				currentPlayer.set_multiplayer_authority(GameManager.Players[i].id)
				currentPlayer.name = str(GameManager.Players[i].id) # can access in c++ with get_
				add_child(currentPlayer)
				Player2 = currentPlayer
				for spawn in get_tree().get_nodes_in_group("SpawnPoint"):
					if spawn.name == "Spawn" + str(index):
						currentPlayer.global_position = spawn.global_position
				index += 1

				for j in GameManager.Players:
					if (j == 1):
						Player2.set_other_id(GameManager.Players[j].id)
					else:
						Player1.set_other_id(GameManager.Players[j].id)
		GameManager.new_client_connected = false
```

Handles disconnection of either player (server or client) midgame and resets for a possible new game.

```gdscript
# called on the server and clients when they disconnect
func peer_disconnected(id):
	print("Player disconnected "  + str(id))
	GameManager.Players.erase(id)
	var players = get_tree().get_nodes_in_group("Players")
	for item in players :
		if item.name == str(id) :
			item.queue_free()
	GameManager.Players = {}
	if (get_node("../main")) :
		get_node("../main").queue_free()
	if GameManager.readied:
		get_tree().change_scene_to_file("res://scenes/disconnect_win_screen.tscn")
	# disconnect signals
	peer = null
	multiplayer.peer_connected.disconnect(peer_connected)
	multiplayer.peer_disconnected.disconnect(peer_disconnected)
	multiplayer.connected_to_server.disconnect(connected_to_server)
	multiplayer.connection_failed.disconnect(connection_failed)
	GameManager.readied = false
```

End conditions and corresponding RPCs to the other player. If one player loses, the other wins, and vice versa. The RPCs handle this.

```cpp
void Player::end_conditions() {
    if (get_position().y < -100.0) {
        game_over = true;
        rpc_id(other_id, "win", "off map other");
    }
    else if (lives < 0) {
        game_over = true;
        tree->change_scene_to_file("res://scenes/no_lives.tscn");
        rpc_id(other_id, "win", "no lives other");
    }
    if (lives == 1) {
        game_over = true;
        tree->change_scene_to_file("res://scenes/ten_lives.tscn");
        rpc_id(other_id, "lose", "ten lives other");
    }
}

void Player::win(String condition) {
    if (condition == "off map other") {
        tree->change_scene_to_file("res://scenes/off_map_other.tscn");
    }
    if (condition == "no lives other") {
        tree->change_scene_to_file("res://scenes/no_lives_other.tscn");
    }
}

void Player::lose(String condition) {
    if (condition == "ten lives other") {
        tree->change_scene_to_file("res://scenes/ten_lives_other.tscn");
    }
}
```

Reconnect to lobby or quit depending on what the player wants to do after the game

```
# restart button
func _on_restart_pressed():
    if GameManager.mode == "multiplayer":
        get_tree().change_scene_to_file("res://scenes/start_menu.tscn")
    else:
        get_tree().change_scene_to_file("res://scenes/main_menu.tscn")

# quit button
func _on_quit_pressed():
    get_tree().quit()
```

The code to implement dead reckoning. The opponent continues to move along their trajectory until they send a new update. There were problems getting the opponent's velocity, so the paths don't currently work.

```
void Player::dead_reckoning(double delta){
    if (sync->get_multiplayer_authority() != get_multiplayer()->get_unique_id()) {
        Vector3 pos = get_global_position();
        Player* other = get_node<Player>("../" + UtilityFunctions::str(get_other_id()));
        set_position(pos + get_local_velocity() * delta);
    }
}
```

What went well and what didn’t

Well:

- We got pretty much all of the multiplayer connections done well, besides connecting / reconnecting while the game was already running. The reason this went well is because there are a large amount of resources on basic godot networking.
- We got the end conditions and their relative GUI scenes done well. This went as expected, with some hiccups along the way but the RPCs helped with that.

Bad

- The dead reckoning code does not work as well as it could due to issues getting velocity data between players. We could move players along a predicted trajectory, but could not get the correct trajectory.
- We had trouble restarting the game after a previous game had ended. We can do this for singleplayer, but after a multiplayer game ends it takes us back to the lobby, and we can’t start a new game.
- Connecting midgame works well on the host side but has a small issue on the client side. The host sees everything updated and completely functional and the client has everything updated and connected except for host movement.