Feature: Player Facing Direction System

Overview

Added a complete player facing/direction system that allows players to turn and face a direction before moving. This creates more tactical gameplay where facing matters, and sets the foundation for directional attacks, sprites, and animations.

Gameplay Behavior

Before: Press arrow key → Player instantly moves to that tile

After:

- Press arrow key while facing different direction → Player turns to face that direction (no movement)
- Press arrow key while already facing that direction → Player moves to that tile

---

Technical Implementation

New Packet Protocol

Opcode	Name	Payload	Direction	Description
`0x01`	Move	`[direction:1][facing:1]`	C → S	Updated to include current facing for server validation
`0x04`	Turn	`[direction:1]`	C → S	New packet for turning without moving
`0x15`	Facing Update	`[facing:1]`	S → C	Server correction/confirmation of facing

Updated Packets (now include facing)

Opcode	Name	New Payload
`0x10`	Your Position	`[x:2][y:2][facing:1]`
`0x12`	Other Player Update	`[id:4][x:2][y:2][facing:1]`
`0x20`	Batch Update	`[count:1][id:4,x:2,y:2,facing:1]...`

Note: Coordinates upgraded from 1 byte to 2 bytes (uint16) to support larger maps up to 65535x65535.

Direction Values

```
0 = Up
1 = Right
2 = Down
3 = Left
```

Server Changes (C++)

Player Class

Added facing state with getter/setter:

```cpp
// Player.h
class Player {
public:
    void SetFacing(uint8_t direction);
    uint8_t GetFacing() const { return facing_; }

private:
    uint8_t facing_ = 2;  // Default: facing down
};
```

PlayerData Struct

```cpp
// Common.h
struct PlayerData {
    uint32_t player_id;
    int x;
    int y;
    uint8_t facing;  // Added
};
```

Packet Helpers

Created `PacketHelpers.h` with reusable read/write functions:

```cpp
```

```
namespace PacketWriter {
    void WriteU8(std::vector<uint8_t>& buffer, uint8_t value);
    void WriteU16(std::vector<uint8_t>& buffer, uint16_t value);
    void WriteU32(std::vector<uint8_t>& buffer, uint32_t value);
}

namespace PacketReader {
    uint8_t ReadU8(const std::vector<uint8_t>& buffer, size_t& offset);
    uint16_t ReadU16(const std::vector<uint8_t>& buffer, size_t& offset);
    uint32_t ReadU32(const std::vector<uint8_t>& buffer, size_t& offset);
}
```

GameSession Packet Handling

Move packet now validates facing before allowing movement:

```cpp
case 0x01: // Move
    if (data.size() >= 3) {
        uint8_t direction = PacketReader::ReadU8(data, offset);
        uint8_t facing = PacketReader::ReadU8(data, offset);

        // Only move if direction matches facing
        if (direction == facing && direction == player_->GetFacing()) {
            bool moved = player_->AttemptMove(direction, server_->GetMap());
            // ...
        } else {
            // Mismatch: turn player, send correction
            player_->SetFacing(direction);
            SendFacingUpdate(player_->GetFacing());
        }
    }
    break;

case 0x04: // Turn
    if (data.size() >= 2) {
        uint8_t direction = PacketReader::ReadU8(data, offset);
        player_->SetFacing(direction);
        is_dirty_ = true;
        SendFacingUpdate(player_->GetFacing());
    }
    break;
```

Client Changes (C#)

NetworkManager State

```csharp
public int MyFacing { get; private set; } = 2;
public Dictionary<uint, int> OtherPlayerFacings { get; private set; } = new();
public System.Action<int> OnMyFacingUpdated;
```

Input Handling

```csharp
private void HandleDirectionInput(int direction) {
    if (MyFacing == direction) {
        SendMoveCommand(direction);   // Already facing, move
    } else {
        SendTurnCommand(direction);   // Turn first
    }
}

public void SendTurnCommand(int direction) {
    byte[] message = new byte[] { 0x04, (byte)direction };
    SendMessage(message);

    // Client-side prediction
    MyFacing = direction;
    OnMyFacingUpdated?.Invoke(MyFacing);
}
```

Packet Helpers

```csharp
```

```
private byte ReadU8(byte[] payload, ref int offset) {
    return payload[offset++];
}

private ushort ReadU16(byte[] payload, ref int offset) {
    ushort value = (ushort)((payload[offset] << 8) | payload[offset + 1]);
    offset += 2;
    return value;
}

private uint ReadU32(byte[] payload, ref int offset) {
    uint value = (uint)((payload[offset] << 24) | (payload[offset + 1] << 16) |
                        (payload[offset + 2] << 8) | payload[offset + 3]);
    offset += 4;
    return value;
}
```

Unity Rendering

GridRenderer Updates

Added sprite array for directional sprites:

```csharp
[SerializeField] private Sprite[] directionSprites;  // 0=up, 1=right, 2=down, 3=left

void Start() {
    // ... existing subscriptions ...
    networkManager.OnMyFacingUpdated += OnMyFacingUpdated;
}

private void OnMyFacingUpdated(int facing) {
    if (localPlayerInstance != null && facing < directionSprites.Length) {
        SpriteRenderer sr = localPlayerInstance.GetComponent<SpriteRenderer>();
        if (sr != null) {
            sr.sprite = directionSprites[facing];
        }
    }
}
```

Setup in Unity Editor

1. Slice sprite sheet with 4 directional sprites

2. On GridRenderer component, set Direction Sprites array size to 4
3. Drag sprites into slots: [0]=Up, [1]=Right, [2]=Down, [3]=Left

---

Files Modified

Server

- `include/server/Player.h` — Added `facing_` member and methods
- `src/server/Player.cpp` — Implemented `SetFacing()`
- `include/server/Common.h` — Added `facing` to `PlayerData` struct
- `include/server/PacketHelpers.h` — **New file** with read/write helpers
- `include/server/GameSession.h` — Added `SendFacingUpdate()` declaration
- `src/server/GameSession.cpp` — Updated packet handling and send functions
- `src/server/GameServer.cpp` — Updated batch packet to include facing

Client

- `Assets/code/NetworkManager.cs` — Added facing state, events, turn command, packet helpers
- `Assets/code/GridRenderer.cs` — Added directional sprite rendering

Testing

1. Start server
2. Connect Unity client
3. Press arrow key in direction not currently facing → Player sprite changes, no movement
4. Press same arrow key again → Player moves in that direction
5. Connect second client → Both players see each other's facing directions