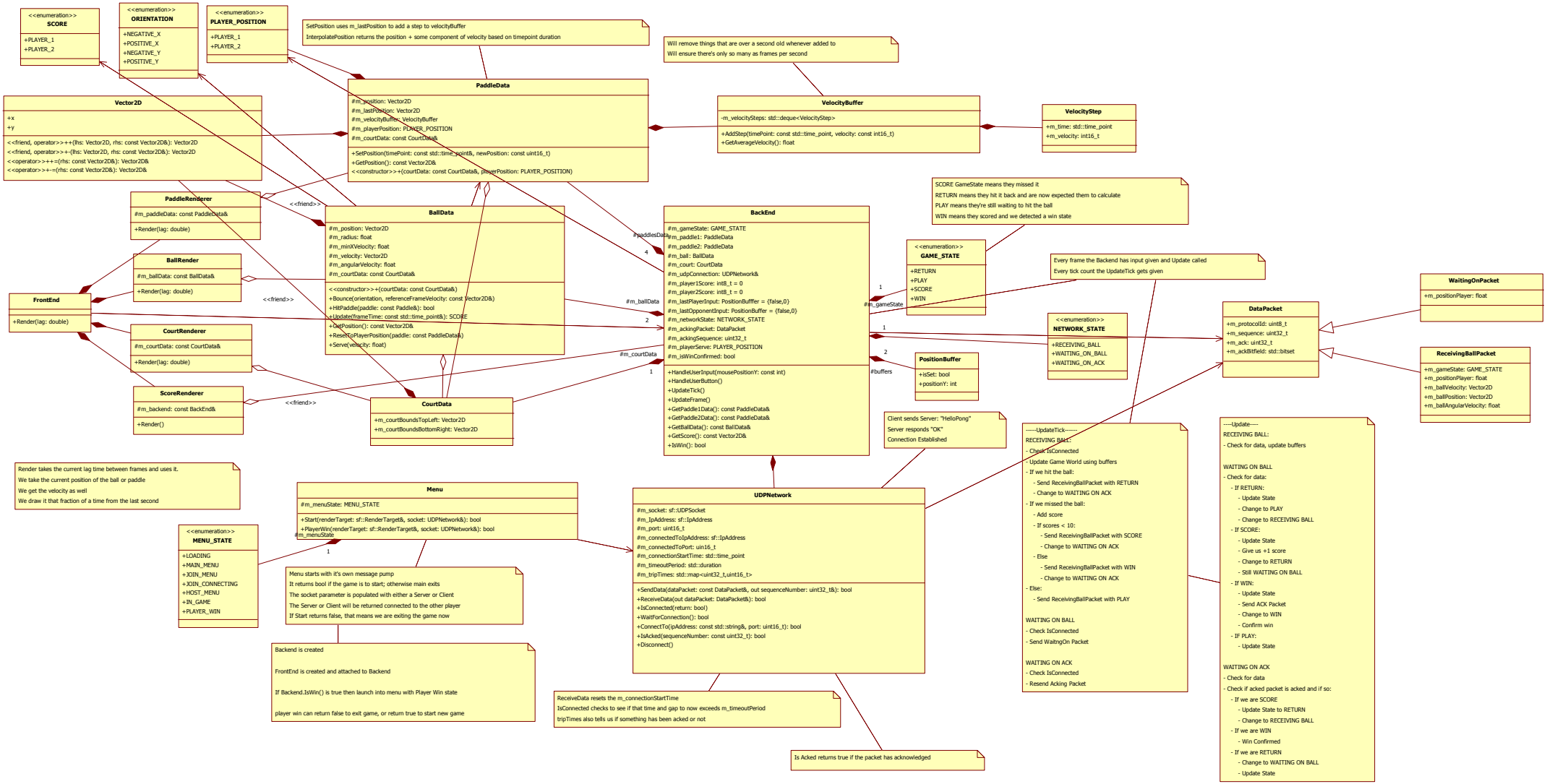
<<enumeration>>
SCORE
+PLAYER_1
+PLAYER_2
<<enumeration>>
ORIENTATION
+NEGATIVE_X
+POSITIVE_X
+NEGATIVE_Y
+POSITIVE_Y
<<enumeration>>
PLAYER_POSITION
+PLAYER_1
+PLAYER_2
SetPosition uses m_lastPosition to add a step to velocityBuffer
InterpolatePosition returns the position + some component of velocity based on timepoint duration
Will remove things that are over a second old whenever added to
Will ensure there's only so many as frames per second
Vector2D
+x
+y
<<friend, operator>>++(lhs: Vector2D, rhs: const Vector2D&): Vector2D
<<friend, operator>>+-(lhs: Vector2D, rhs: const Vector2D&): Vector2D
<<operator>>+=(rhs: const Vector2D&): Vector2D&
<<operator>>+-=(rhs: const Vector2D&): Vector2D&
PaddleData
#m_position: Vector2D
#m_lastPosition: Vector2D
#m_velocityBuffer: VelocityBuffer
#m_playerPosition: PLAYER_POSITION
#m_courtData: const CourtData&
+SetPosition(timePoint: const std::time_point&, newPosition: const uint16_t)
+GetPosition(): const Vector2D&
<<constructor>>+(courtData: const CourtData&, playerPosition: PLAYER_POSITION)
VelocityBuffer
-m_velocitySteps: std::deque<VelocityStep>
+AddStep(timePoint: const std::time_point, velocity: const int16_t)
+GetAverageVelocity(): float
VelocityStep
+m_time: std::time_point
+m_velocity: int16_t
PaddleRenderer
#m_paddleData: const PaddleData&
+Render(lag: double)
<<friend>>
BallData
#m_position: Vector2D
#m_radius: float
#m_minXVelocity: float
#m_velocity: Vector2D
#m_angularVelocity: float
#m_courtData: const CourtData&
<<constructor>>+(courtData: const CourtData&)
+Bounce(orientation, referenceFrameVelocity: const Vector2D&)
+HitPaddle(paddle: const Paddle&): bool
+Update(frameTime: const std::time_point&): SCORE
+GetPosition(): const Vector2D&
+ResetToPlayerPosition(paddle: const PaddleData&)
+Serve(velocity: float)
#paddlesData
4
#m_ballData
2
#m_courtData
1
BackEnd
#m_gameState: GAME_STATE
#m_paddle1: PaddleData
#m_paddle2: PaddleData
#m_ball: BallData
#m_court: CourtData
#m_udpConnection: UDPNetwork&
#m_player1Score: int8_t = 0
#m_player2Score: int8_t = 0
#m_lastPlayerInput: PositionBufffer = {false,0}
#m_lastOpponentInput: PositionBuffer = {false,0}
#m_networkState: NETWORK_STATE
#m_ackingPacket: DataPacket
#m_ackingSequence: uint32_t
#m_playerServe: PLAYER_POSITION
#m_isWinConfirmed: bool
+HandleUserInput(mousePositionY: const int)
+HandleUserButton()
+UpdateTick()
+UpdateFrame()
+GetPaddle1Data(): const PaddleData&
+GetPaddle2Data(): const PaddleData&
+GetBallData(): const BallData&
+GetScore(): const Vector2D&
+IsWin(): bool
SCORE GameState means they missed it
RETURN means they hit it back and are now expected them to calculate
PLAY means they're still waiting to hit the ball
WIN means they scored and we detected a win state
<<enumeration>>
GAME_STATE
+RETURN
+PLAY
+SCORE
+WIN
1
#m_gameState
Every frame the Backend has input given and Update called
Every tick count the UpdateTick gets given
BallRender
#m_ballData: const BallData&
+Render(lag: double)
FrontEnd
+Render(lag: double)
<<friend>>
CourtRenderer
#m_courtData: const CourtData&
+Render(lag: double)
ScoreRenderer
#m_backend: const BackEnd&
+Render()
<<friend>>
CourtData
+m_courtBoundsTopLeft: Vector2D
+m_courtBoundsBottomRight: Vector2D
<<enumeration>>
NETWORK_STATE
+RECEIVING_BALL
+WAITING_ON_BALL
+WAITING_ON_ACK
1
2
#buffers
PositionBuffer
+isSet: bool
+positionY: int
DataPacket
+m_protocolId: uint8_t
+m_sequence: uint32_t
+m_ack: uint32_t
+m_ackBitfield: std::bitset
WaitingOnPacket
+m_positionPlayer: float
ReceivingBallPacket
+m_gameState: GAME_STATE
+m_positionPlayer: float
+m_ballVelocity: Vector2D
+m_ballPosition: Vector2D
+m_ballAngularVelocity: float
Client sends Server: "HelloPong"
Server responds "OK"
Connection Established
Render takes the current lag time between frames and uses it.
We take the current position of the ball or paddle
We get the velocity as well
We draw it that fraction of a time from the last second
Menu
#m_menuState: MENU_STATE
+Start(renderTarget: sf::RenderTarget&, socket: UDPNetwork&): bool
+PlayerWin(renderTarget: sf::RenderTarget&, socket: UDPNetwork&): bool
#m_menuState
1
<<enumeration>>
MENU_STATE
+LOADING
+MAIN_MENU
+JOIN_MENU
+JOIN_CONNECTING
+HOST_MENU
+IN_GAME
+PLAYER_WIN
Menu starts with it's own message pump
It returns bool if the game is to start; otherwise main exits
The socket parameter is populated with either a Server or Client
The Server or Client will be returned connected to the other player
If Start returns false, that means we are exiting the game now
Backend is created
FrontEnd is created and attached to Backend
If Backend.IsWin() is true then launch into menu with Player Win state
player win can return false to exit game, or return true to start new game
UDPNetwork
#m_socket: sf::UDPSocket
#m_IpAddress: sf::IpAddress
#m_port: uint16_t
#m_connectedToIpAddress: sf::IpAddress
#m_connectedToPort: uin16_t
#m_connectionStartTime: std::time_point
#m_timeoutPeriod: std::duration
#m_tripTimes: std::map<uint32_t,uint16_t>
+SendData(dataPacket: const DataPacket&, out sequenceNumber: uint32_t&): bool
+ReceiveData(out dataPacket: DataPacket&): bool
+IsConnected(return: bool)
+WaitForConnection(): bool
+ConnectTo(ipAddress: const std::string&, port: uint16_t): bool
+IsAcked(sequenceNumber: const uint32_t): bool
+Disconnect()
ReceiveData resets the m_connectionStartTime
IsConnected checks to see if that time and gap to now exceeds m_timeoutPeriod
tripTimes also tells us if something has been acked or not
Is Acked returns true if the packet has acknowledged
-----UpdateTick-----
RECEIVING BALL:
- Check IsConnected
- Update Game World using buffers
- If we hit the ball:
  - Send ReceivingBallPacket with RETURN
  - Change to WAITING ON ACK
- If we missed the ball:
  - Add score
  - If scores < 10:
    - Send ReceivingBallPacket with SCORE
    - Change to WAITING ON ACK
  - Else
    - Send ReceivingBallPacket with WIN
    - Change to WAITING ON ACK
- Else:
  - Send ReceivingBallPacket with PLAY
WAITING ON BALL
- Check IsConnected
- Send WaitngOn Packet
WAITING ON ACK
- Check IsConnected
- Resend Acking Packet
----Update----
RECEIVING BALL:
- Check for data, update buffers
WAITING ON BALL
- Check for data:
  - If RETURN:
    - Update State
    - Change to PLAY
    - Change to RECEIVING BALL
  - If SCORE:
    - Update State
    - Give us +1 score
    - Change to RETURN
    - Still WAITING ON BALL
  - If WIN:
    - Update State
    - Send ACK Packet
    - Change to WIN
    - Confirm win
  - IF PLAY:
    - Update State
WAITING ON ACK
- Check for data
- Check if acked packet is acked and if so:
  - If we are SCORE
    - Update State to RETURN
    - Change to RECEIVING BALL
  - If we are WIN
    - Win Confirmed
  - If we are RETURN
    - Change to WAITING ON BALL
    - Update State