Technical Design Document

CRC Cards

Packet	
Responsibilities	Collaborators
Appends the past in data into a buffer of integer values in preparation to be sent across the network.	PlayerData EndData StartData

PlayerData	
Responsibilities	Collaborators
Stores, sets and returns the index of the player and the velocity of the player.	

StartData	
Responsibilities	Collaborators
Stores, sets and returns the index of the player and the index of the target player.	

EndData	
Responsibilities	Collaborators
Stores, sets and returns the index of the player, the index of the target player and the time the game has lasted for.	

ChangeState	
Responsibilities	Collaborators
Manages the enum Gamestate that is used to create. It is able to convert itself along with its contents into a packet and return the pointer to the created packet.	Packet

GameStart

Technical Design Document

Responsibilities	Collaborators
Manages the StartData that is used to create this class. It is able to convert itself along with its contents into a packet and return the pointer to the created packet.	Packet StartData

PlayerUpdate	
Responsibilities	Collaborators
Manages the PlayerData that is used to create this class. It is able to convert itself along with its contents into a packet and return the pointer to the created packet.	Packet PlayerData

GameUpdate	
Responsibilities	Collaborators
Manages an array of vector positions that is used to create this class. It is able to convert itself along with its contents into a packet and return the pointer to the created packet.	Packet

GameEnd	
Responsibilities	Collaborators
Manages the EndData that is used to create this class. It is able to convert itself along with its contents into a packet and return the pointer to the created packet.	Packet EndData

PacketManager	
Responsibilities	Collaborators
Manages all the packets that are stored within its internal queue. It is able to clear the queue, add and retrieve the first element within the queue of Packets	Packet

Client	
Responsibilities	Collaborators

Technical Design Document

The client upon start tries to establish connection with the passed in ip address on the passed in port. The Client is able to request from Server to connect to one of its sockets after which it creates two threads which are used for send and listening for Packets respectively. The client is able to close up the connection with the server and merge packet the two threads. The Client is able to derence the received packets into their original structure and call an instance of game to deal with the received data. The client is also able to send the received data from the game to the server after it converts it to a packet.	Packet Game PacketType

Server	
Responsibilities	Collaborators
The server creates listening sockets on the port that listen for either local or public client connections that it refers to as Connections. After the server is created it listens for new client connections adding them to a list of established connections as long as the active connection count has not been exceeded. The server is able to receive the data and unpack it from Connection and send it to an instance of the Game to deal with it appropriately as long as there are pending Packets from a Connection. The Server is able to send data received from Game to all active Connections. The Server is able to remove a Connection and clean it up if the Connection is lost to that Client.	Game Connection PacketType PlayerData StartData EndData

Connection	
Responsibilities	Collaborators
Stores the socket which was used to create this instance of Connection. Stores the id of the Connection.	PacketManager

Technical Design Document

Manages the PacketManager which stores all the data from a Client that has been received on this socket.	

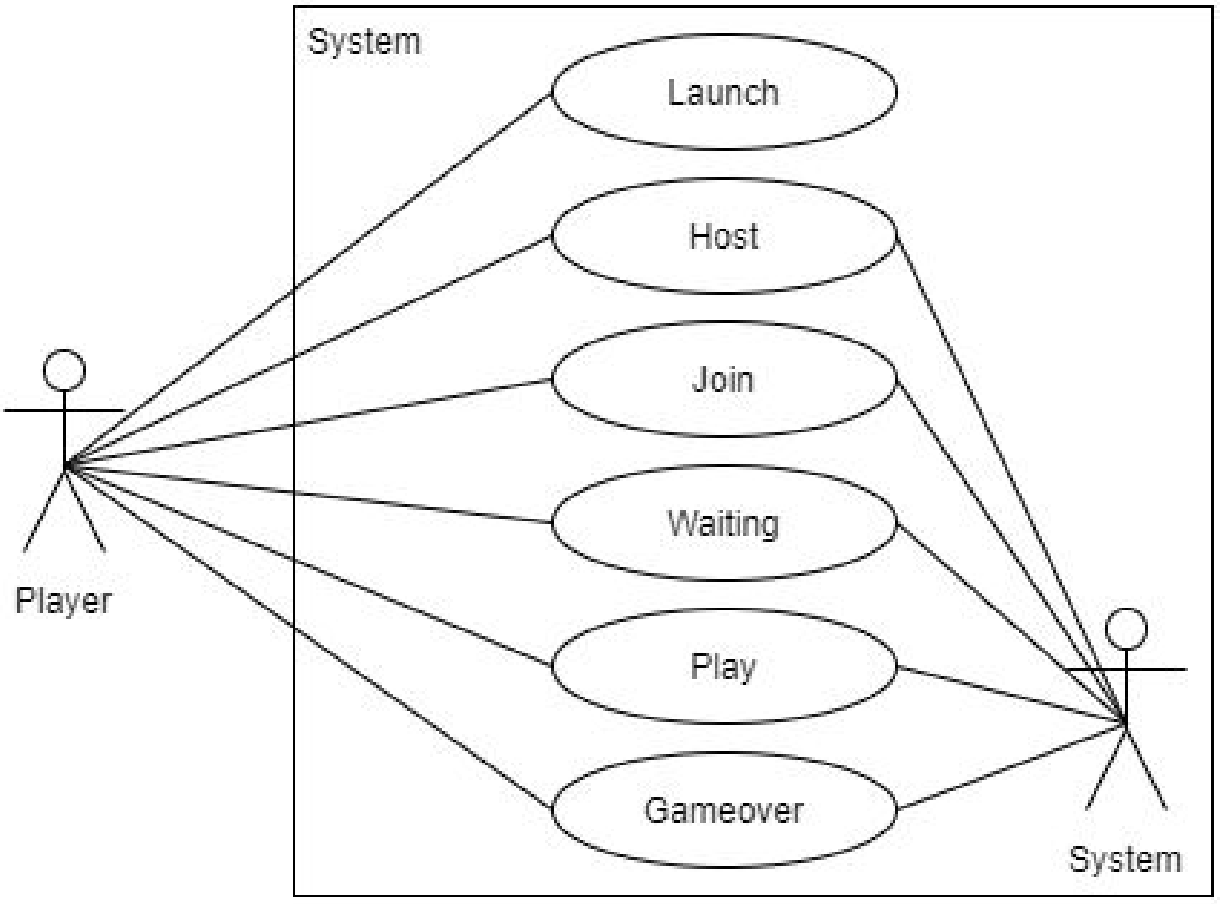
Game	
Responsibilities	Collaborators
Controls the game loop of this project. Creates and updates the render and input window of the application. Creates, updates and resets the instances of Player that it contains within a vector. Creates and updates all the UI elements that are used by this application. Creates and stores an instance of Client or Server depending on which the Player decided that they wish to be. Sends and processes the data to/from Client/Server depending on what the user has selected. Changes the state the program is in based on the internal or external events. Checks for collision between the Player and responds accordingly. Creates a thread for a Server to listen for new connections if a user chooses to be host.	Player Server Client

Player	
Responsibilities	Collaborators
Stores the position of the Player within the Game world. Draws the body of the Player on screen. Calculates the velocity of the Player based on the active inputs each frame and returns it. Stores the name of the Player. Calculates if the position of the Player is out of bounds and deals with it accordingly.	VectorMath

VectorMath	
Responsibilities	Collaborators
Computes vector calculations on the passed in vectors to each of the functions.	

Technical Design Document

Use Case Diagram:



Use Cases:

Name: Launch
Actor: Player
Description: This use case begins when the player launches the game. The game then displays two buttons one saying “Host” one saying “Join”. This use case ends when the Player clicks either two of these buttons.

Name: Host
Actor: Player, System
Description: This use case begins when the Player presses the host button during launch during this time the user will see two more buttons appear asking ”Do you want to broadcast publicly” with one button saying “Yes” and the other “No”. When the Player presses on either of the buttons the system creates a server with the appropriate setting and the use case ends going into Waiting.

Name: Join
Actor: Player, System
Description: This use case begins when the Player presses the join button during launch the Player can see text “Enter the ip of the host” where the Player can enter the ip of the host he wants to connect to. This use case ends when the Player presses the “Confirm”

button during which time the System creates a Client and attempts to connect to the server using the entered ip address.

Name: Waiting
Actor: Player, System
Description: This use case begins when the Player either completes the Host or Join use case successfully or a player drops out during the other use cases. During this time the Player with the server is waiting until two more Clients connect. This use case ends when two more Clients connect during which time the system gets the server to change the state of the server and the two other clients to Play.

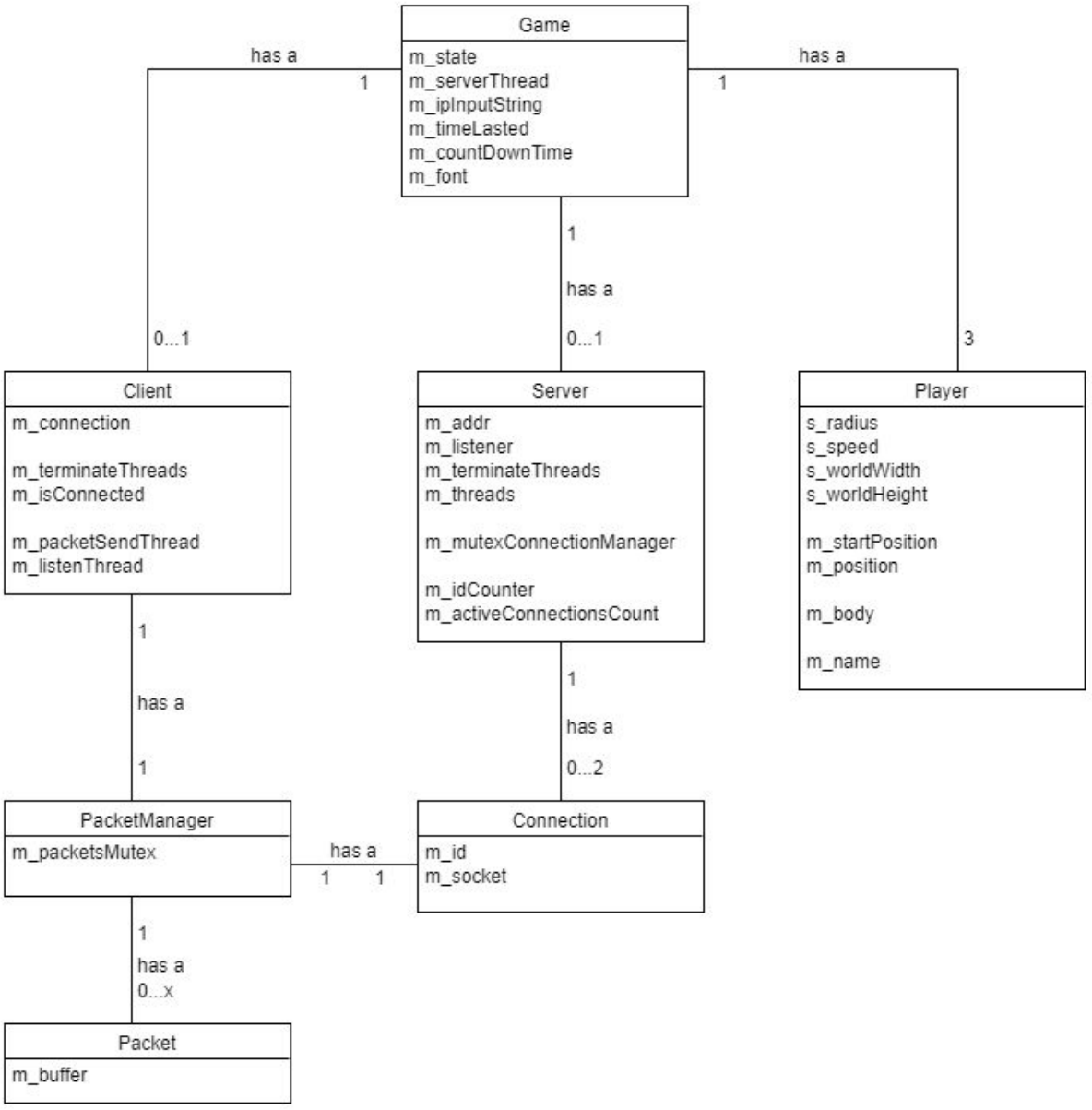
Name: Play
Actor: Player, System
Description: This use case begins once the Waiting state completes. At the start of this state each Player is able to see which character they are on screen and which character they need to catch. The Player can use the arrow keys to move around with their character. This use case ends when any of the Players disconnects or the character that is chasing the target collides with any of the other characters.

Name: Gameover
Actor: Player, System
Description: This use case begins when the Play use case concludes successfully during which the system displays the name of the Player that has caught the chased Player and how long the chased Player lasted. The fail state version of this use case begins when the Server closes for the Clients in this case the message is displayed “The server has shutdown”. This use case ends when the Player closes down the application.

Domain Model:

Classes that have been omitted within this diagram are used to send the data across and thus exist for only a limited time within the project thus they were not displayed within the domain model.

Technical Design Document



Technology:

For this project I will be using visual studio 2019 as the editor for the code using c++. This is due to the specificationtions of the project where we are required to use WINSOCK as the network plugin.

As mentioned above for the networking part I will be using the Winsock extensions for the creation of both the server and lenient components of the project and I will utilize its function in order to bind the sockets and send data across as an integers.

For the rendering of the project I will try to utilize SFML library in order to render and position all the UI and game objects on screen. I will be using SFML as I am most familiar with it for c++ projects and the specifications of the projects fit the capabilities of the project quite well.