V6

1	2	3
4	5	6
7	8	9

| 1 byte | 1 byte | 1 byte | 1 byte | 1 byte | 1 byte | 1 byte

Maximum size: 1000 bytes

Useful Size: 7 bytes

All values are unsigned

Field descriptions:

- First byte: Version (6)
- Second byte: Position [1,9]
- Third byte: Game state [0,2]
    - 2 == General error
    - 1 == Game complete
    - 0 == Game in progress
- Fourth byte: Modifier of the third byte [1,5]
    - Case: General Error
        - 1 == Out of resources (can't accept a new game at this time)
        - 2 == Malformed/invalid request
        - 3 == Server shutdown
        - 4 == Client game timeout (server to client)
        - 5 == Try again
    - Case: Game complete
        - 1 == Draw

            - 2 == Client wins
            - 3 == Server wins
        - Case: Game in progress
            - No Info Flag
    - Fifth Byte: Command [0,2]
        - 0 == New game
        - 1 == Move
        - 2 == End game
    - Sixth Byte: Game Number, indicates what game is being played [0, 255]
        - Assigned by server
    - Seventh Byte: Sequence Number [0,255]
        - Wrapped back to 0
        - Increment by 1
        - Can start anywhere (initiated by client)
        - Sequence is shared between client and server (ex: actor sends message with sequence 1 and expects next message received to have sequence 2)
    - All undefined bytes reserved for future use, can be considered as junk

Initial Handshake:

- To start a game there must be a handshake procedure:
    1. Client sends command “new game”, version number as first byte, starting sequence number as seventh byte, other fields irrelevant
    2. Server Responds with game number, which will be used by both parties to identify future move, or general error, with error field set appropriately
    3. Game board is still blank, after receiving game number client makes first move

Normal Play:

- ·Run after initial handshake:
    1. Client sends move to server with command field set to move (1), and game number set to the clients game number, with end game fields set appropriately
    2. Server responds with move, sets ‘end game’ field appropriately

- If the client sends a ‘new game’ request in the middle of a running game:
    1. Server sends ‘general error’ with a ‘try again’ error code, and ends current game
    2. Client can retry ‘new game’ request

Packet Acknowledgement:

- When a packet is resent, for the purpose of this lab, it will not be possible for that packet to contain different information than the original
- Three retries possible. When timeout occurs, send a retry. After three retries, we timeout the entire game.

Ending Handshake:

(Final Move)

First Byte = Version

Second Byte = Last move

Third Byte = Game Complete

Fourth Byte = Appropriate (Client wins/Server wins/Draw)

Fifth Byte (Command) = Move

Sixth Byte = Game Number

Seventh Byte = Sequence Number (incremented)

(Response)

First Byte = Version

Third Byte = Game Complete

Fourth Byte = Appropriate (Client wins/Server wins/Draw)

Fifth Byte (Command) = End Game

Sixth Byte = Game Number

Seventh Byte = Sequence Number (incremented)

Waits timeout length to verify there is no resend request before exiting.

MAX GAMES: 10

Notes:

- Client plays first
- User-chosen `timeout
- Nothing is an ASCII value, example for all values: (1 -> 0b00000001)
- If possible to set the error message, it is strongly recommended to do so

These are not protocol these are error checking on your programs/additional info:

- Stdint.h has the typedefs
  https://pubs.opengroup.org/onlinepubs/009696799/basedefs/stdint.h.html
  - <inttypes.h> has definitions for printf on top of everything stdint.h includes
  - https://stackoverflow.com/questions/7597025/difference-between-stdint-h-and-inttypes-h