This document outlines the specification for the Pacman Protocol, intended for use in a two-player online Pacman game. The document defines the game's communication protocol, including the message contents, formats, timing, and encoding methods.

Student ID： 24012383

Pacman Protocol Specification
===

Terminology
============

This specification uses the terms MUST, SHOULD, and MAY as defined in RFC 2119 [rfc2119].

The Pacman protocol runs over **both TCP and UDP**, using a well-known port 1234.
There are 12 message types:

MAZE_UPDATE

PACMAN_ARRIVED
PACMAN_LEFT
PACMAN_GO_HOME
PACMAN_DIED

FOREIGN_PACMAN_ATE_GHOST
EAT

SCORE_UPDATE
LIVES_UPDATE
STATUS_UPDATE

PACMAN_UPDATE
GHOST_UPDATE

The **MAZE_UPDATE** message is critical for initializing the game setup, so it MUST be sent using **TCP** protocol to ensure data is not lost or corrupted.

PACMAN_UPDATE and **GHOST_UPDATE** messages are sent at a high frequency, their function is to continuously update the game visuals, the occasional loss of a message won't have a significant impact on the game, after all the players are unlikely to notice that. So, they SHOULD be sent using **UDP** to minimize latency and overhead.

Meanwhile, the other nine messages, which correspond to events in the game process, such as LIVES_UPDATE, SCORE_UPDATE are also vital for the players' experience, so they SHOULD be sent via **TCP** protocol to ensure these messages are reliably captured and processed when corresponding events occur.

The solution for handling possible UDP messages out-of-order, and UDP messages may overtake TCP ones, will be discussed at the end.

Message Timing
==============

The MAZE_UPDATE message is sent at the beginning of the game to initialize maze information for two computers.

PACMAN_UPDATE and GHOST_UPDATE need to be sent frequently to maintain smooth gameplay and ensure the positions of the Pacman and ghosts are updated promptly. Considering the network speed and computational overhead, these two messages should be sent **20 times per second**, corresponding to sending every 50 milliseconds.

The other seven messages are sent when the corresponding events occur. For instance, SCORE_UPDATE is sent when the score changes, LIVES_UPDATE when Pacman loses a life, and PACMAN_ARRIVED when Pacman reaches another map.

Message Encoding
================

Messages are in fixed format, each message follows a specific fixed structure where the fields have a consistent position and order, ensuring uniformity and predictability in data representation.

Messages are encoded directly into binary format rather than in human-readable text formats such as JSON or XML, optimizing for efficiency and compactness. This also allows for easier parsing and serialization/deserialization of messages.

All floating-point values, such as the x and y coordinates and speed of Pacman and ghosts, need to be represented in the IEEE 754 standard format as single-precision floating-point numbers, which will take up 32 bytes.

All integer fields in all messages are sent in a network byte order -- big-endian order which the most significant byte is at the beginning of the byte array.

Here is an example of a 4-byte integer 0x04321567
- Big-endian: 0x04 0x32 0x15 0x67
- Little-endian: 0x67 0x15 0x32 0x04

As a message type is fixed forward, no explicit length field is required. Multiple messages MAY be transmitted sequentially within a single packet. This approach can be advantageous in minimizing overhead, particularly when dispatching both PACMAN_UPDATE and GHOST_UPDATE messages together.

To ensure the sanity of received values, the receiver MUST check the values through the validation standards outlined below:

- Message Type Validation:
Authenticate the first 4 bits field of the message, this is an integer representing the message type, and it MUST fall within the range of 0 to 11.

- Field Length Validation:
Verify that each message has the correct length. For instance, the PACMAN_UPDATE message always has 16 bytes, SCORE_UPDATE always has 4 bytes, LIFE_UPDATE always has 1 byte, and so on.
If the length of the received message is not equal to the expected length, the message MUST be discarded.

- Enumeration Validation:
Verify that enumerated values are within the expected set. For instance, the direction of the Pacman and ghosts MUST fall within the range of 0 to 4. The mode of the ghost MUST fall within the range of 0 to 1, and the status of the game MUST fall within the range of 0 to 5, and so on.
If the values are outside the predefined range, the message MUST be discarded.

- Range Validation:
Verify that the values, such as coordinates and speeds of the Pacman and ghosts, adhere to the range defined in advance in the file. For instance, the x coordinates should be within the range of 0 to CANVAS_WIDTH = 650, the y coordinates should be within the range of 0 to CANVAS_HEIGHT = 800.
If the values are outside the predefined range, the message MUST be discarded.

Determining Mazes
=================

At the start stage of the game, the mazes of two computers MUST be determined. Two players can choose their own LOCAL mazes by typing "-m <mazenum>" on the command line, mazenum is an integer from 0 to 2. If no corresponding command input is provided, the maze will be randomly selected. Once the LOCAL maze is determined, a copy of LOCAL maze will be sent to the other player's computer using the MAZE_UPDATE message, this maze will be displayed as a "REMOTE" maze in the upper right.

This message MUST be transmitted reliably and without any packet loss to both computers to successfully synchronize the mazes information.
Hence, this transmission process MUST be done over TCP.

Message Contents and Format
============================

MAZE_UPDATE content and format

Type: MAZE_UPDATE, mapped to a decimal value of 0.
Content: ["maze", maze], a list containing the string "maze" and a class named "maze" which contains maze information. It includes:
- Maze level: an integer from 0 to 2, indicating the shape of the maze
- Maze shape: which is stored in the file "maze0.txt", "maze1.txt" and "maze2.txt" that are in the current path.
- Current Level: The difficulty of the game.
- Walls
- Tunnels
- Total number of foods

MAZE_UPDATE messages consist of **23492 bytes**, encoded as follows:

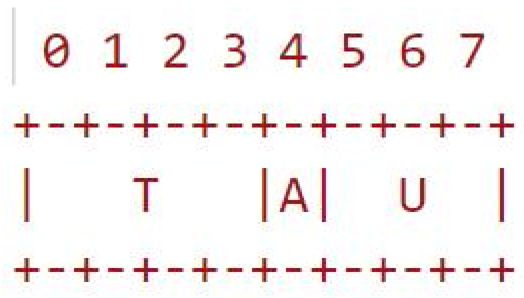
Fields:
- Type: 4 bits field. Type = MAZE_UPDATE has a decimal value of 0.
- Level: an integer from 0 to 2, indicating the shape of the maze. 4 bits unsigned integer in big-endian byte order.
- Shape: 18432 bits unsigned integer in big-endian byte order.
- CL: the difficulty of the game.
  4 bits unsigned integer in big-endian byte order.
- Wall: 3472 bits unsigned integer in big-endian byte order.
- Tunnel: 128 bits unsigned integer in big-endian byte order.
- Food: 4 bits unsigned integer in big-endian byte order.
- Unused: 4 bits, not used, but needed to maintain byte alignment.
  MUST be set to zero in this version of the protocol.

Protocol Type: TCP

PACMAN_ARRIVED content and format

Type: PACMAN_ARRIVED, mapped to a decimal value of 1.
Content: A Boolean value indicating whether the Pacman has arrived at another maze.
0 for False, not arrived; 1 for True, arrived.

PACMAN_ARRIVED messages consist of **1 byte**, encoded as follows:



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   |A|  U  |
+-+-+-+-+-+-+-+-+
```

Fields:

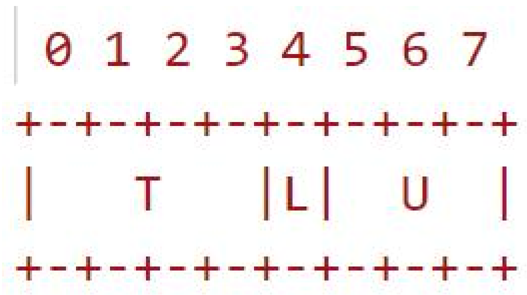
- T: 4 bits field. Type = PACMAN_ARRIVED has a decimal value of 1.
- A: 1 bit for indicating whether the Pacman has arrived
  (0 for False, not arrived; 1 for True, arrived)
- U: 3 bits, not used, but needed to maintain byte alignment.
  MUST be set to zero in this version of the protocol.

Protocol Type: TCP

PACMAN_LEFT content and format
--

Type:	PACMAN_LEFT, mapped to a decimal value of 2.
Content:	A Boolean value indicating whether the Pacman has left the current maze. 0 for False, not left; 1 for True, left.

PACMAN_LEFT messages consist of **1 byte**, encoded as follows:



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   |L|  U  |
+-+-+-+-+-+-+-+-+
```

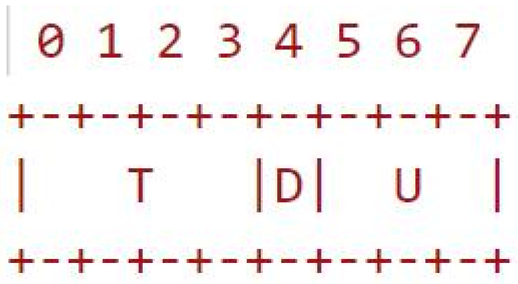
Fields:
- T: 4 bits field. Type = PACMAN_LEFT has a decimal value of 2.
- L: 1 bit for indicating whether the Pacman has left
  (0 for False, not left; 1 for True, left)
- U: 3 bits, not used, but needed to maintain byte alignment.
  MUST be set to zero in this version of the protocol.

Protocol Type: TCP

PACMAN_DIED content and format
--

Type:	PACMAN_DIED, mapped to a decimal value of 3.
Content:	A Boolean value indicating whether the Pacman has died. 0 for False, not died; 1 for True, died.

PACMAN_DIED messages consist of 1 byte, encoded as follows:



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   |D|  U  |
+-+-+-+-+-+-+-+-+
```

Fields:

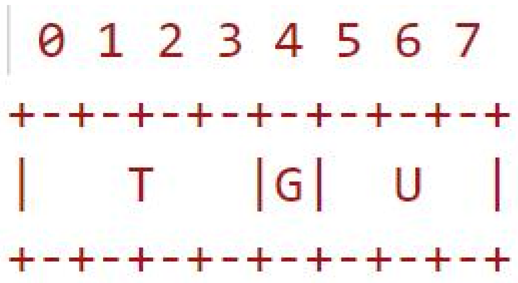
- T:	4 bits field. Type = PACMAN_DIED has a decimal value of 3.
- D:	1 bit for indicating whether Pacman has died (0 for False, not died; 1 for True, died)
- U:	3 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

Protocol Type: TCP

PACMAN_GO_HOME content and format
--

Type:	PACMAN_GO_HOME, mapped to a decimal value of 4.
Content:	A Boolean value indicating whether the Pacman has gone home. 0 for False, not gone home; 1 for True, gone home.

PACMAN_GO_HOME messages consist of **1 byte**, encoded as follows:



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   |G|  U  |
+-+-+-+-+-+-+-+-+
```

Fields:

- T:	4 bits field. Type = PACMAN_GO_HOME has a decimal value of 4.
- G:	1 bit for indicating whether the Pacman has gone home

(0 for False, not gone home; 1 for True, gone home)

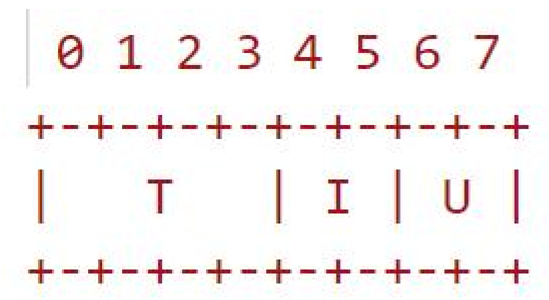
- U: 3 bits, not used, but needed to maintain byte alignment.
  MUST be set to zero in this version of the protocol.

Protocol Type: TCP

FOREIGN_PACMAN_ATE_GHOST content and format

Type:	FOREIGN_PACMAN_ATE_GHOST, mapped to a decimal value of 5.
Content:	After Pacman eats a power pill, the game mode will be changed to FRIGHTEN. In this mode, Pacman can eat ghosts. This message contains an integer from 0 to 3 represents the ID of the ghost that was eaten by Pacman.

FOREIGN_PACMAN_ATE_GHOST messages consist of **1 byte**, encoded as follows:



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   | I | U |
+-+-+-+-+-+-+-+-+
```

Fields:

- T: 4 bits field. Type = FOREIGN_PACMAN_ATE_GHOST has a decimal value of 5.
- I: 2 bits to represent decimal unsigned integers from 0 to 3 in big-endian byte order
- U: 2 bits, not used, but needed to maintain byte alignment.
  MUST be set to zero in this version of the protocol.

Protocol Type: TCP

EAT content and format

Type: EAT, mapped to a decimal value of 6.

Content:

- a list containing the x and y coordinates of the food that is eaten by Pacman.
  These two values are float type.
- a Boolean value indicating whether the food is foreign or not.
- a Boolean value indicating whether the food is a powerpill or not.

EAT messages consist of **9 bytes**, encoded as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   T   |F|P| U |               Food x-coord                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   x-(cont)    |               Food y-coord                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   y-(cont)    |
+-+-+-+-+-+-+-+-+
```

Fields:

- T: 4 bits field. Type = EAT has a decimal value of 7.
- Food x-coord: 32 bits float in big-endian byte order.
- Food y-coord: 32 bits float in big-endian byte order.
- F: 1 bit for indicating whether the food is foreign or not.
                        (0 for False, not foreign; 1 for True, foreign)
- P: 1 bit for indicating whether the food is a powerpill or not.
                        (0 for False, not powerful; 1 for True, powerpill)
- U: 6 bits, not used, but needed to maintain byte alignment.

Protocol Type: TCP

SCORE_UPDATE content and format

Type: SCORE_UPDATE, mapped to a decimal value of 7.
Content: an integer represents the score of the current game.

SCORE_UPDATE messages consist of **4 bytes**, encoded as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   T   |                     Score                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

- T: 4 bits field. Type = SCORE_UPDATE has a decimal value of 7.
- Score: 28 bits unsigned integer in big-endian byte order to represent the score of the current game, the maximum value is 2^28=268,435,455.

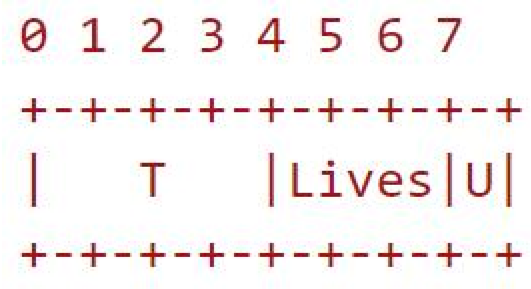
Protocol Type: TCP

LIVES_UPDATE content and format

--

Type:	LIVES_UPDATE, mapped to a decimal value of 8.
Content:	an integer represents the remaining lives of the Pacman.

LIVES_UPDATE messages consist of **1 byte**, encoded as follows:



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   |Lives|U|
+-+-+-+-+-+-+-+-+
```

Fields:

- T: 4 bits field. Type = LIVES_UPDATE has a decimal value of 8.
- Lives: 3 bits unsigned integer in big-endian byte order to represent the remaining lives of the Pacman, maximum value is 2^3=8. But in this game, the maximum value is 5.
- U: 1 bit, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

Protocol Type: TCP

STATUS_UPDATE content and format

Type:	STATUS_UPDATE, mapped to a decimal value of 9.
Content:	an integer represents the status of the game.
	- 0: STARTUP
	- 1: CHASE
	- 2: FRIGHTEN
	- 3: GAVE_OVER
	- 4: NEXT_LEVEL_WAIT
	- 5: READY_TO_RESTART

STATUS_UPDATE messages consist of **1 byte**, encoded as follows:

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|   T   |  S  |U|
+-+-+-+-+-+-+-+-+
```

Fields:

- T:	4 bits field. Type = STATUS_UPDATE has a decimal value of 9.
- S:	3 bits unsigned integer in big-endian byte order to represent the status of the game.
- U:	1 bit, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

Protocol Type: TCP

PACMAN_UPDATE content and format

Type: PACMAN_UPDATE, mapped to a decimal value of 10.

Content: - a list containing the x and y coordinates of the Pacman's current position in the current maze. These two values are float type.

- an integer indicating the current direction of the Pacman.
  - 0: UP
  - 1: LEFT
  - 2: RIGHT
  - 3: DOWN
  - 4: NONE

- a float indicating the current speed of the Pacman.

This is a sample of the appearance of a message before it is encoded:

['pacman', [(162.44941916574737, 340), <Direction.LEFT: 1>, 1]]

PACMAN_UPDATE messages consist of **16 bytes**, encoded as follows:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   T   | Dir |U|              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Pacman x-coord                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Pacman y-coord                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Pacman speed                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

- T: 4 bits field. Type = PACMAN_UPDATE has a decimal value of 10.
- Dir: 3 bits unsigned integer in big-endian byte order to represent the current direction of the Pacman.
- U: 1 bit, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.
- Sequence Number: 24 bits unsigned integer in big-endian byte order. Used to ensure the order of the messages.
- Pacman x-coord: 32 bits float in big-endian byte order.
- Pacman y-coord: 32 bits float in big-endian byte order.
- Pacman speed: 32 bits float in big-endian byte order.

Protocol Type: UDP

This is what the message looks like after encoding, based on the example above:
0x0A 0x01 0x00 0x03001D05 0x42A506D0 0x43700000 0x3F800000

GHOST_UPDATE content and format
--

Type: GHOST_UPDATE, mapped to a decimal value of 11.

Content:
- an integer indicating the ID of the ghost.
- a list containing the x and y coordinates of the ghost's current position in the current maze. These two values are float type.
- an integer indicating the current direction of the ghost.
    - 0: UP
    - 1: LEFT
    - 2: RIGHT
    - 3: DOWN
    - 4: NONE
- a float indicating the current speed of the ghost.
- an integer indicating the mode of the ghost.
    - 0: FRIGHTEN
    - 1: CHASE

This is a sample of the appearance of a message before it is encoded:

['ghost', [0, (280.0, 228.6373600951738), <Direction.UP: 0>, 0.9, <GhostMode.CHASE: 1>]]

GHOST_UPDATE messages consist of **16 bytes**, encoded as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   T   | Dir |M|              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Ghost x-coord                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Ghost y-coord                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Ghost speed                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

- T:	4 bits field. Type = GHOST_UPDATE has a decimal value of 11.
- Dir:	3 bits unsigned integer in big-endian byte order to represent the current direction of the ghost.
- M:	1 bit for indicating the mode of the ghost.

	(0 for FRIGHTEN; 1 for CHASE)
- Sequence Number:	24 bits unsigned integer in big-endian byte order. Used to ensure the order of the messages.
- Ghost x-coord:	32 bits float in big-endian byte order.
- Ghost y-coord:	32 bits float in big-endian byte order.
- Ghost speed:	32 bits float in big-endian byte order.

- Protocol Type: UDP

This is what the message looks like after encoding, based on the example above:
0x0B 0x00 0x01 0x0100140A 0x43A80000 0x43D8F5C3 0x3F4CCCCD

Sequence Number | About TCP & UDP

==============================

Due to the use of UDP protocol, PACMAN_UPDATE and GHOST_UPDATE messages, may be lost or arrive out of order. If earlier messages arrive later than more recent messages, it will cause issues with position updates and animations.
To handle the issue of out-of-order messages, a sequence number is added to each PACMAN_UPDATE and GHOST_UPDATE message.
This sequence number, a 24-bit number like a timestamp that is incremented by 1 for each message sent, logs the time when the message was sent. The receiver SHOULD establish a buffer to compare the sequence number of the received message with the previous ones.
If the latest one is smaller than any of the previous ones stored in the buffer, the message MUST be discarded.

Another issue to consider is the difference in arrival speed between UDP and TCP messages.
A UDP message might arrive before a TCP message even though they were sent at the same time.
For instance, Pacman might already have reached a position where a food item exists, but this food is not displayed until after Pacman has passed it.
To avoid this issue, a buffer SHOULD be established on the receiver side. UDP messages SHOULD be deferred for processing, such as with a 5-millisecond delay, rather than being processed immediately upon arrival.
During this delay, the receiver SHOULD check for any incoming TCP messages. If any are received, they should be processed first, followed by the UDP messages.