Pacman Protocol Specification

Terminology

This specification uses the terms MUST, SHOULD, and MAY as defined in the RFC 2119 [rfc2219]

The Pacman protocol runs over both TCP and UDP, using a well known port of 5432.
There are eleven message types:

1. maze_update
2. pacman_arrived
3. pacman_left
4. pacman_died
5. pacman_go_home
6. pacman_update
7. ghost_update
8. ghost_eaten
9. eat
10. score_update
11. status_update

The local model MUST know everything about the remote model when the pacman is away. Therefore, at the beginning of each game, both computers send a copy of their maze to each other in a maze_update message. The maze that is sent contains the positions of all the food and powerpills on the maze.

Everytime the local pacman visits the remote maze, the local computer sends a pacman_arrived message to the remote computer. The computer that received the message uses this message to initialize any state.

Whenever an away pacman comes back to the local maze, pacman_left message is sent by the local model. This will inform the remote computer that the pacman left that maze; therefore, the remote side will stop displaying that pacman on its main maze. Additionally, if the pacman is forcibly sent home, the position of the pacman will be reset and the local model will send pacman_left message.

If the local model recognizes that its away pacman was killed by a remote ghost, then pacman_died message will be sent by the local computer.

In some cases, the away pacman is directly sent back to its home. During this process, the remote system sends a pacman_go_home message. Then the local system resets the pacman's position and the pacman returns to the local maze. Examples of situations that requires this message: the level is completed, one pacman dies, etc.

Whenever a pacman moves, the local computer sends a pacman_update message to the remote computer. The message is independent from the current position of the pacman. It can either be local or away. In the pacman_update message, remote computer receives the current position, direction, and speed of the pacman.

Whenever a local ghost moves, the local computer sends a ghost_update message to the remote computer. This message includes, the index, position, speed, direction, and mode of the ghost.

Whenever the local model detects that an away pacman has eaten a remote ghost, it sends a ghost_eaten message to the remote system. Of course, the ghost MUST be in frighten mode to be eaten.

Eat message is sent by the local system to remote computer if the pacman (local or away) eats a food or a powerpill. This message is used for informing the remote system about the new update.

Each computer MUST be able to know the score for each user. Therefore, the local computer sends a score_update message to the remote computer whenever a pacman's score alters. The score_update message is sent to the other computer whether the pacman is local or away.

The local game board has some different states associated with it. These states are STARTUP, CHASE, FRIGHTEN, GAME_OVER, NEXT_LEVEL_WAIT, and READY_TO_RESTART. The current state of the game is sent by using the status_update message.

Determining Mazes

Before starting the game, mazes of two computers MUST be determined. Users can choose their own maze choice by typing it on command line while running the game. If they do not specify a maze, the maze will be selected randomly. Three possible mazes have different index numbers (0,1, and 2). The game automatically selects a random number between 0 and 2 in order to choose the maze. All mazes' X and Y values are between 0-1023.

- Type: maze_update
- Contents: The outline of the local maze is sent to the remote computer to allow it to visualize the shape of the local maze. Also, the maze that is received by the remote model includes X and Y values of all the food and powerpills to determine the location of them on the maze. After receiving this message, the remote computer displays the local maze on top right of the screen.

As TCP is a reliable protocol, the information about the maze will be sent to the remote computer by using TCP. Both users MUST be able to see both mazes fully; therefore, a missing data can make the game unplayable.

Message Contents
==========

The contents of a pacman_arrived message are:

- Type: pacman_arrived
- Value: This message only sends a number between 0-1 to notify the remote computer when the event in the message occurs.

The contents of a pacman_left message are:

- Type: pacman_left
- Value: This message only sends a number between 0-1 to notify the remote computer when the event in the message occurs.

The contents of a pacman_died message are:

- Type: pacman_died
- Value: This message only sends a number between 0-1 to notify the remote computer when the event in the message occurs.

The contents of a pacman_go_home message are:

- Type: pacman_go_home
- Value: This message only sends a number between 0-1 to notify the remote computer when the event in the message occurs.

The contents of a pacman_update message are:

- Type: pacman_update
- Value: Position of the pacman in pixels in order to understand the point where the pacman is located at on the maze. This message also contains the current direction and speed of the pacman.

The contents of a ghost_update message are:

- Type: ghost_update
- Value: Each ghost has a different ghost number which allows the game to distinguish them. This message contains the ghost number to determine the ghost that this message updates. It also contains the current position of the specified ghost. Moreover,

current speed and direction is also sent in this message. Plus, the mode of the ghost is also mentioned in this message. It shows whether the ghost is in chase or frighten mode.

The contents of a ghost_eaten message are:

- Type: ghost_eaten
- Value: This message only sends a number between 0-1 to notify the remote computer when the event in the message occurs.

The contents of a eat message are:

- Type: eat
- Value: This message contains the position of the food and the information of whether the eaten food was foreign. Also, it sends whether the food was a powerpill or not.

The contents of a score_update message are:

- Type: score_update
- Value: This message only sends the current score.

The contents of a status_update message are:

- Type: status_upgrade
- Value: This message only sends the the current status of the game.

Message Timing
==========
While the pacman is moving, pacman update message SHOULD be sent every 20ms, since this is the average reaction time of humans. If a computer can not maintain 50 frames per second, pacman update messages can be lowered to once per frame, since it is illogical to update the remote computer more often than the local computer.

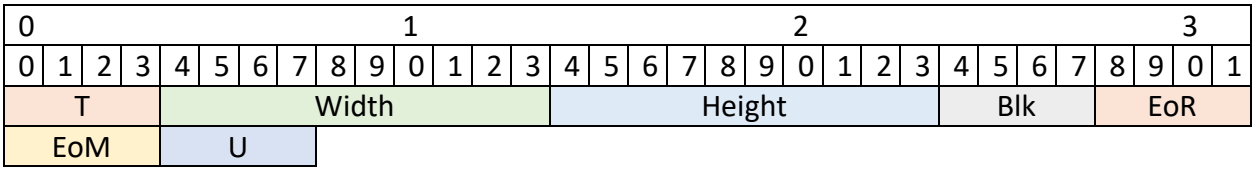
Similarly, the ghost update message SHOULD be sent every 20ms, as it is the typical reaction time for humans. Again, if a computer can not maintain this, once per frame would be enough.

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

All messages are binary encoded with all integer fields send in network byte order (ie. Big endian order). As message type is fixed forward, no explicit length field is required. More than one message MAY be sent consecutively in a single packet.

maze_update message format
---

maze_update messages consist of 5 bytes, encoded as follow:



T: 4 bit type field

Width: 10 bit unsigned integer in big-endian byte order

Height: 10 bit unsigned integer in big-endian byte order

Blk: 4 bit unsigned integer in big-endian byte order

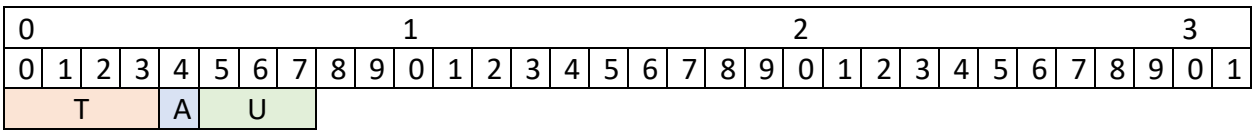
EoR: 4 bit unsigned integer in big-endian byte order

EoM: 4 bit unsigned integer in big-endian byte order

U: 4 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

pacman_arrived message format
---

pacman_arrived message consist of 1 byte, encoded as follows:

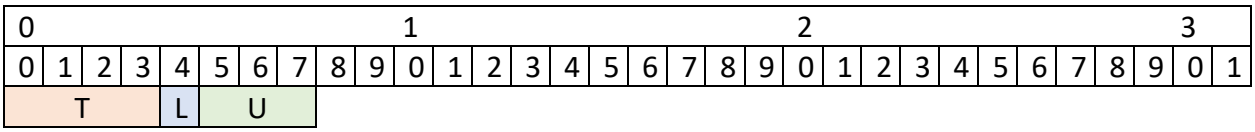


T: 4 bit type field.

A: 1 bit for notifying whether the pacman arrived or not (0 for no, 1 for arrived).

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

pacman_left message format
---
pacman_left message consists of 1 byte, encoded as follows:

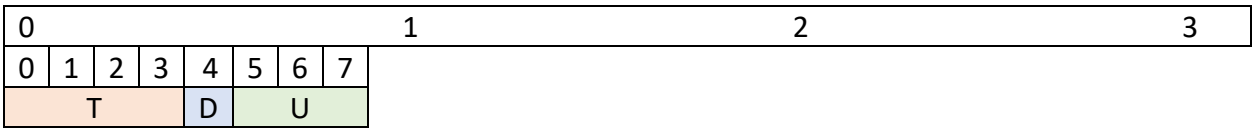


T: 4 bit type field.

L: 1 bit for notifying whether the pacman left or not (0 for no, 1 for left).

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

pacman_died message format
---
pacman_died message consists of 1 byte, encoded as follows:

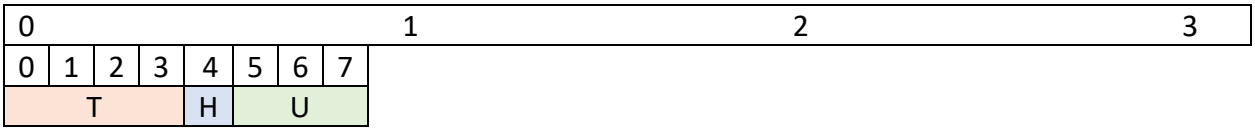


T: 4 bit type field.

D: 1 bit for notifying whether the pacman died or not (0 for no, 1 for died).

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

pacman_go_home message format
---
pacman_go_home message consists of 1 bye, encoded as follows:

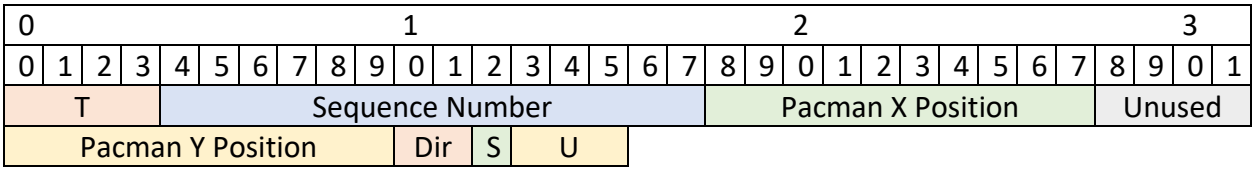


T: 4 bit type field

H: 1 bit for notifying whether the pacman went to home or not (0 for no, 1 for yes).

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

pacman_update message format
---
pacman_update message consists of 6 bytes, encoded as follows:



T: 4 bit type field.

Sequence Number: A 14 bit unsigned integer, incremented by one for every new message sent. If it reaches $2^{14} - 1$, it wraps back round to zero.

Pacman X Position, Pacman Y Position: 10 bits, giving an unsigned integer in big-endian byte order.

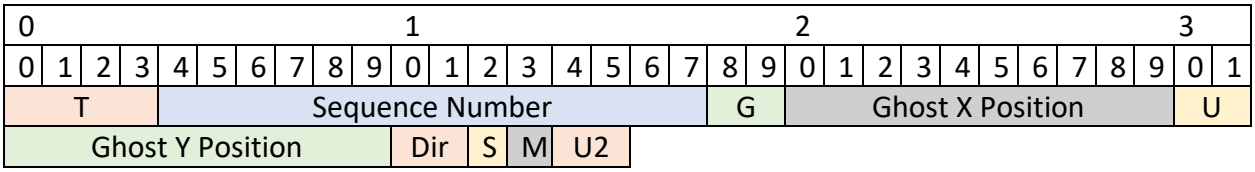
Unused: 4 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

Dir: 2 bits for sending the direction of the pacman (00, 01, 10, 11)

S: 1 bit for sending the speed of the pacman (either 0 or 1).

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

ghost_update message format
---
ghost_update message consists of 6 bytes, encoded as follows:



T: 4 bit type field

Sequence Number: A 14 bit unsigned integer, incremented by one for every new message sent. If it reaches $2^{14} - 1$, it wraps back round to zero.

G: 2 bits for sending the index of the ghost (00, 01, 10, 11).

Ghost X Position, Ghost Y Position: 10 bits, giving an unsigned integer 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.

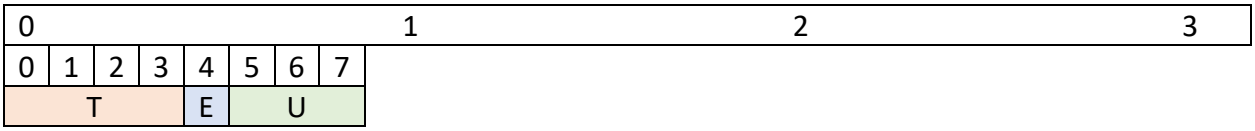
Dir: 2 bits for sending the direction of the pacman (00,01,10,11)

S: 1 bit for sending the speed of the pacman (either 0 or 1).

M: 1 bit for sending the mode of the pacman (0 for chase, 1 for frighten).

U2: 2 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

ghost_eaten message format
---
ghost_eaten message consists of 1 byte, encoded as follows:

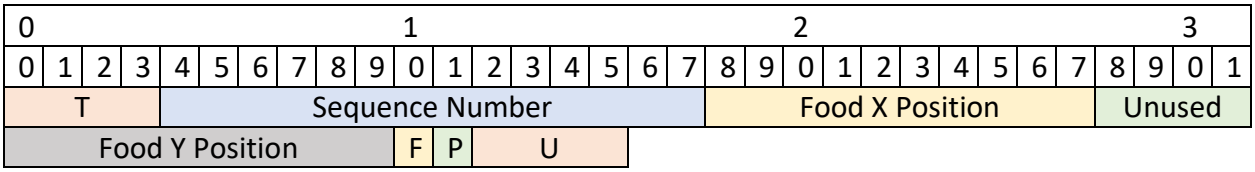


T: 4 bit type field

H: 1 bit for notifying whether the pacman ate a ghost or not (0 for no, 1 for yes).

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

eat message format
---
eat message consists of 6 bytes, encoded as follows:



T: 4 bit type field

Sequence Number: A 14 bit unsigned integer, incremented by one for every new message sent. If it reaches $2^{14} - 1$, it wraps back round to zero.

Food X Position, Food Y Position: 10 bits, giving an unsigned integer in big-endian byte order.

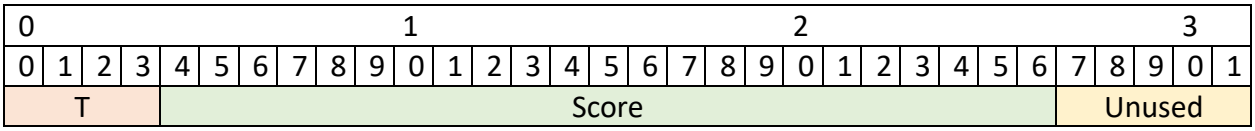
Unused: 4 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

F: 1 bit for determining whether the food was foreign or not (0 for local, 1 for away).

P: 1 bit for determining whether the food was a powerpill (0 for not, 1 for powerpill).

U: 4 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

score_update message format
---
score_update message consists of 4 bytes, encoded as follows:

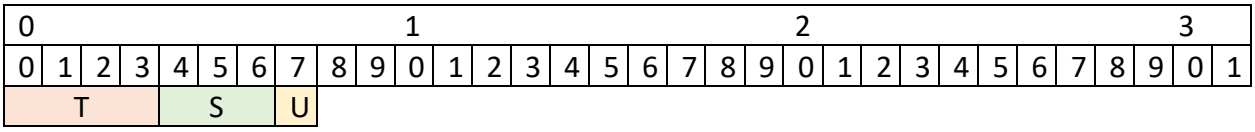


T: 4 bit type field

Score: 23 bit unsigned integer to send the current score. The maximum score that can be sent is $2^{23} - 1$.

Unused: 5 bits, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

status_update message format
---
status_update message consists of 1 byte, encoded as follows:



T: 4 bit type field

S: 3 bit for determining the states

U: 1 bit, not used, but needed to maintain byte alignment. MUST be set to zero in this version of the protocol.

Sequence Numbers

Due to the use of UDP, messages may be lost or arrive out of order. We do not want to update the screen with out of order data. The receiver therefore keeps track of the sequence number of the last message received of each type. If it receives a message of a specific type with a lower sequence number than the last one received, the message MUST be discarded. When performing this comparison, care must be taken to account for the potential for sequence numbers to wrap.