Student Number: 21014779

Pacman Protocol Specification

Abstract

The purpose of this report on the multi-player Pacman game is to specify the method of communication between server and client about game state. In particular it will specify the messages to be sent and the conditions that must be satisfied or intervals in order for them to be sent.

Terminology

This document uses words such as MUST, SHOULD and MAY which are referenced from the RFCs 2119 document.

The game session will be managed in Peer-to-peer network. Each node can act as client and server as in this case they are not distinguished. Also, it is a network within which each user has its own data and applications which is appropriate for the Pacman game. The client MUST start the session every-time to avoid delays and problems. A client to server to client model might be considered however, implementing one would create potentially greater and unfair latency. Sanity checks to prevent exploits on the server could be useful however the game is aimed at friends/peers who will hopefully be a good sport.

The Pac-Man protocol runs over TCP, using well known port of 5432. Transmission Control Protocol (TCP) is a communication protocol. It is chosen in particular due to its guaranteed packet order and delivery. These features help prevent race conditions and allow easier debugging and maintenance of code. This improves the correctness of the code and ease of implementation.
There are twelve message types are: MAZE_UPDATE, PACMAN_ARRIVED, PACMAN_LEFT, PACMAN_DIED, PACMAN_GO_HOME, PACMAN_UPDATE, GHOST_UPDATE, GHOST_WAS_EATEN, EAT, SCORE_UPDATE and STATUS_UPDATE.

We use the following terminology in this specification to distinguish between visiting Pacman and other game objects:

1. LOCAL: The game object presently on the local screen.
2. AWAY: A pacman currently away on remote screen.
3. REMOTE: a game object on the remote screen that our AWAY pacman might interact with.
4. FOREIGN: The other player's pacman, when it is visiting our screen.
5. REPLICA: The other player's local game board that has been replicated onto our screen.

In this document, when these terms are capitalized, they have these specific meanings.

Message timings

The following outline of message timings assumes that pacman is being played such that two game boards can be seen per person. Updates to the visual REPLICA game board can simply be ignored by the program if only one game board is being displayed to each person. However, states of the REPLICA and the other player's pacman information are still tracked regardless of mode.

When our pacman is AWAY, the local model needs to know about everything it can interact with. **At game start or restart, each computer sends the other a copy of its maze with a "maze update" message.** The game ships with three different mazes, though more can be added. Mazes are a fixed size of 28 x 29 squares (filled with vertical wall, horizontal wall, top right corner, top left corner, bottom right corner, bottom left corner, empty, food, powerpill, inaccessible, tunnel A, tunnel B). A player can choose a maze by selecting "-m " on the command line, where mazenum is an integer, typically from 0 to 2. If no maze is specified, it is selected randomly. The maze that is sent includes the location of all the food and powerpills.

The model running on your computer keeps two mazes in memory - the LOCAL one and the REMOTE one.

To keep the copies of the maze running on your computer and the remote computer synchronized, each computer continuously informs the other of actions.

On receipt of these messages from the remote computer, the local computer updates the relevant maze - this might be the LOCAL maze if the other pacman is currently FOREIGN (ie it is visiting our maze), or it might be the REPLICA maze if the other pacman is REMOTE.

Whenever our pacman or our LOCAL ghosts move (every frame), our computer sends "pacman update" or "ghost update" messages to the remote computer, giving the current position, direction of our pacman and ghosts. Messages are sent irrespective of whether our pacman is LOCAL or AWAY. If our pacman is AWAY, the remote computer updates its foreign pacman. If our pacman is LOCAL, the remote computer updates our pacman in the replica maze. **If the last update was less than 20 ms ago, then a message is not sentas rates faster than 20 ms are not particularly noticeable to players.**

Similarly, our computer also accepts "pacman update" and "ghost update" messages to update either the local or replica game board with pacman and the ghosts positions accordingly.

The local game board also has states associated with it. These are defined in the class GameMode in pa model.py:

- STARTUP
- CHASE
- FRIGHTEN
- GAME OVER
- NEXT LEVEL WAIT
- READY TO RESTART

All changes between these states on the local game board are communicated to the remote computer using "status update" messages.

During the intervals between "pacman update" and "ghost update" messages to the remote computer, the remote computer locally linearly interpolates the pacmen and ghost positions of its local models using the known speed and direction of the pacmen and ghosts respectively. The speed of pacman is known as it is constant. The speed of ghosts is known as it is also constant depending on whether the ghost is in "FRIGHTEN" mode or "CHASE" mode. In "Chase" mode their speed is 0.8 except for one ghost whose speed is 0.9 (not sure if this was intentional), In "FRIGHTEN" mode, their speed is half of their "CHASE" mode speed. This interpolation helps ease unpredictable delays of TCP messaging.

Following position updates, based on our computer's local model, irrespective of whether our pacman is LOCAL or AWAY, our computer determines any collisions by using either the replica or LOCAL board. **If our pacman eats food or a powerpill, it sends an "eat" message to the remote computer irrespective of whether the ghost eaten is LOCAL to our computer or the remote computer. The receiving computer works out if food or a powerpill was eaten based on the position of the object eaten. The score is then incremented or the state is changed to "Frighten" on the correct board. The game then checks to see if there is any food left on the board it was eaten from, if there is none the game changes to the NEXT LEVEL WAIT state.**

The states of both the LOCAL and REPLICA game board are stored locally. This prevents unnecessary passing of ghost state in every "ghost update" message. Only the state change message is passed and the LOCAL system remembers the current state of the REPLICA board. When the "FRIGHTEN" mode initialises, the frightened values of all ghosts in the board are set to true. Any checks of whether ghosts are frightened when our pacman is AWAY read from the locally stored state of the ghosts in the REPLICA board.

A "pacman died" message is sent if our pacman is killed by ghosts. If our model detects that our pacman has eaten a ghost (while it was in FRIGHTEN mode), it sends a "ghost was eaten" message to the remote computer irrespective of whether the ghost eaten is LOCAL to our computer or the remote computer. Upon receipt of this message, the system respawns the ghost as an unfrightened ghost and listens for further instruction on the ghosts movements if the ghost is REMOTE.

Some events require that our AWAY pacman be forcibly sent home. **This happens when the level is completed on the remote screen. The remote computer sends a "pacman go home" message. Our system then resets our pacman to LOCAL, and sends a "foreign pacman left" message in reply.**

Distinguishments between which boards' ghosts, food and powerpills are eaten are made using message contents. Our pacman keeps track of its own score and lives, updating the remote model's information about our pacman's score and lives with the "pacman died" message (the remote model decrements lives each time it receives this message and resets lives when necessary) and "score update".

Whether or not our pacman will traverse a tunnel is also handled by the computer owning pacman. **If our pacman will move to the remote game board to become AWAY, our computer first sends a "pacman arrived" message, so the remote computer can place**

the pacman on its game board and initialise any state. If our pacman leaves the remote game board to become LOCAL, our computer sends a "foreign pacman left" message to the remote computer. The remote computer upon receiving this message stops displaying the foreign pacman.

Thus, even when our pacman is AWAY, interactions between it and food, ghosts, tunnels and powerpills are still handled by the local model. Its score and lives are also handled by the local model as a result.

There is one race condition scenario which inflates the score:

- LOCAL Pacman and FOREIGN Pacman start in the same position on the same board
- They move toward the same Direction and start to eat the food before it is deleted.

If this happens, each client will think that their Pacman ate the food and increase their scores, effectively doubling the food they eat. **However, as Mark has said, there is no need to implement a mechanism to fix this.** In practice one way to fix it might be to have both computers start a precise timer and communicate the timestamps that they eat the food to see who eats what first.

The gameplay outlined above only happens in CHASE and FRIGHTEN state (the difference being whether a powerpill has recently been eaten). The software is in STARTUP state while playing the startup jingle.

If either player loses their last life, the game ends. The losing player's computer goes to GAME OVER state, and sends a status update message. The other side then also moves to GAME OVER state.

From GAME OVER state, if the local player presses "r" to restart, the local computer goes to READY TO RESTART state and sends an update message. The game restarts when the second player also presses "r", and sends a replying "READY TO RESTART" status update.

When a level is cleared on a screen by eating all food, that screen's system goes to NEXT LEVEL WAIT while it plays the jingle and the player gets ready. Completing a level does not affect the level being played on the other screen, except the pacmen positions are reset if they are on the game board that is completed.

The complete list of messages in the current version of the protocol is therefore:

1. maze update
2. pacman arrived
3. foreign pacman left
4. pacman died
5. pacman go home
6. pacman update
7. ghost update
8. ghost was eaten
9. eat
10. score update
11. status update

Message content

==========

The contents of a MAZE_UPDATE message are:

- Type: MAZE_UPDATE

- Value: The contents of each square in the maze, sequentially from (1,1), (2,1),(3,1) where (x,y) x is the row number starting from 1 at the top and y is the column number starting from 1 at the left. The contents of each square is represented by a character (a, b, c, d, e, f, g, h, i, j, k, l) corresponding to: vertical wall, horizontal wall, top right corner, top left corner, bottom right corner, bottom left corner, empty, food, powerpill, inaccessible, tunnel A, tunnel B respectively.

The contents of a PACMAN_ARRIVED message are:

- Type: PACMAN_ARRIVED

- Value: X, Y is the position of the pacman. X is distance across the canvas from 20 to 520 as pacman moves from left to right from the point of view of Player 1. Y is distance along the screen from 20 to 580 where (20, 20) is the top left of the available canvas and (520, 580) is the bottom left of the available canvas. The direction has 4 possibilities (left, right, up, down) represented by (1, 2, 3, 4) respectively.

The contents of a FOREIGN_PACMAN_LEFT message are:

- Type: FOREIGN_PACMAN_LEFT

- Value: No value, since this message is only sent when the pacman needs to leave the remote screen.

The contents of a PACMAN_DIED message are:

- Type: PACMAN_DIED

- Value: No value, since this message is only sent when the pacman has died.

The contents of a PACMAN_GO_HOME message are:

- Type: PACMAN_GO_HOME

- Value: No value, since this message is only sent when the pacman goes home.

The contents of a PACMAN_UPDATE message are:

- Type: PACMAN_UPDATE

- Value: X, Y is the position of the pacman. X is distance across the canvas from 20 to 520 as pacman moves from left to right from the point of view of Player 1. Y is

distance along the screen from 20 to 580 where (20, 20) is the top left of the available canvas and (520, 580) is the bottom left of the available canvas. The direction has 4 possibilities (left, right, up, down) represented by (1, 2, 3, 4) respectively.

The contents of a GHOST_UPDATE message are:

- Type: GHOST_UPDATE

- Value: X, Y is the position of the pacman. X is distance across the canvas from 20 to 520 as the ghost moves from left to right from the point of view of Player 1. Y is distance along the screen from 20 to 580 where (20, 20) is the top left of the available canvas and (520, 580) is the bottom left of the available canvas. The ghost numbers are within 1 to 4 (n1, n2, n3, n4). The information of all the ghosts is sent all together in one GHOST_UPDATE message.

The contents of a GHOST_WAS_EATEN message are:

- Type: GHOST_WAS_EATEN

- Value: An integer value of 1 to 4, to indicate which ghost is eaten. An integer of 0 or 1 to indicate whether the ghost eaten was on the LOCAL or REMOTE board of the computer which sent the message.

The contents of an EAT message are:

- Type: EAT

- Value: X, Y square position of the food. X is row number of the food unit. Y is column number of the food unit. They range from 1 to 28 and 1 to 29 respectively, where (1, 1) is the top left square and (28, 29) is the bottom right square. An integer of 0 or 1 is sent to indicate whether the food/powerpill eaten was on the LOCAL or REMOTE board of the computer which sent the message.

The contents of a SCORE_UPDATE message are:

- Type: SCORE_UPDATE

- Value: An integer value in the range 0 to 9999.

The contents of a STATUS_UPDATE message are:

- Type: STATUS_UPDATE

- Value: State of the game, represented by (1, 2, 3, 4, 5, 6) for the modes STARTUP, CHASE, FRIGHTEN, GAME OVER, NEXT LEVEL WAIT, READY TO RESTART respectively

Message encoding

==========

Text encoding is used as it is more developer friendly and readable. It is much easier to find bugs and eases implementation, improving correctness. The messages are in fixed format, ASCII encoded, separated by a newline character to distinguish messages.

CR characters MUST NOT be sent. More than one message MAY be sent consecutively in a single packet- this may be useful to reduce overhead when sending both PACMAN_UPDATE and GHOST_UPDATE.

SCORE_UPDATE message format

SCORE_UPDATE messages are of variable length encoded as follows:

SCORE: Score=<score><newline>

“SCORE" MUST be capitalized, and MUST be at the start of the connection or immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<score> is a decimal ASCII encoding of the round number, and MUST be in the range 0 to 9999.

EAT message format

EAT messages are of variable length encoded as follows:

EAT: XPosition=<food_x_position> YPosition=<food_y_position>
Local=<local><newline>

“EAT” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<food_x_position> and <food_y_position> are decimal ASCII encodings of x and y positions of the food, and MUST be in the range 1 to 28 and 1 to 29 respectively as there are 28x29 squares.

MAZE_UPDATE message format

MAZE_UPDATE messages are of variable length encoded as follows:

MAZE_UPDATE: One=<one>
two=<two>...EightHundredTwelve=<eight_hundred_twelve><newline>

“MAZE_UPDATE” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<one> … <eight_hundred_twelve> are ASCII encodings of the contents of each square of the maze sequentially from (1,1), (2,1),(3,1) where (x,y) x is the row number starting from 1 at the top and y is the column number starting from 1 at the left. They MUST be one of the characters a, b, c, d, e, f, g, h, i, j, k, l.

PACMAN_ARRIVED message format

PACMAN_ARRIVED messages are of variable length encoded as follows:

PACMAN_ARRIVED: XPos=<X_Position_Pacman> YPos=<Y_Position_Pacman> Direction=<Direction><newline>

“PACMAN_ARRIVED” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<X_Position_Pacman> and <Y_Position_Pacman> are decimal ASCII encodings of the position of the Pacman and MUST be in the range 20 to 520 and 20 to 580 respectively. <Direction> is a ASCII encoding of the direction of the Pacman and MUST be in the range 1 to 4.

FOREIGN_PACMAN_LEFT message format

FOREIGN_PACMAN_LEFT messages are of variable length encoded as follows:

FOREIGN_PACMAN_LEFT: <newline>

“FOREIGN_PACMAN_LEFT” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

PACMAN_DIED message format

PACMAN_DIED messages are of variable length encoded as follows:

PACMAN_DIED: <newline>

“PACMAN_DIED” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

PACMAN_GO_HOME message format

PACMAN_GO_HOME messages are of variable length encoded as follows:

PACMAN_GO_HOME: <newline>

“PACMAN_GO_HOME” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

PACMAN_UPDATE message format

PACMAN_UPDATE messages are of variable length encoded as follows:

PACMAN_UPDATE: XPos=<X_Position_Pacman> YPos=<Y_Position_Pacman> Direction=<Direction><newline>

“PACMAN_UPDATE” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<X_Position_Pacman> and <Y_Position_Pacman> are decimal ASCII encodings of the position of the Pacman and MUST be in the range 20 to 520 and 20 to 580 respectively.
<Direction> is a ASCII encoding of the direction of the Pacman and MUST be in the range 1 to 4.

GHOST_UPDATE message format

GHOST_UPDATE messages are of variable length encoded as follows:

GHOST_UPDATE: GhostNum= <Ghost_Number>(every ghost number 1 to 4) XPos=<Ghost_X_Pos> (every ghost X position 20 to 520) YPos=<Ghost_Y_Pos>(every ghost Y position 20 to 580) <newline>

“GHOST_UPDATE” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<Ghost_Number><Ghost_X_Pos><Ghost_Y_Pos> are decimal ASCII encodings of the ghosts’ position and number.
<Ghost_Number> must be between 1 to 4 inclusive.
<Ghost_X_Pos> and <Ghost_Y_Pos> MUST be between 20 to 520 and 20 to 580 respectively.

GHOST_WAS_EATEN message format

GHOST_WAS_EATEN messages are of variable length encoded as follows:

GHOST_WAS_EATEN: GhostNum= <ghost_num> Local=<local><newline>

“GHOST_WAS_EATEN” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<ghost_num> is a decimal ASCII encoding which MUST be in the range 1 to 4.
<local> is a decimal ASCII encoding which MUST be in the range 0 to 1.

STATUS_UPDATE message format

STATUS_UPDATE messages are of variable length encoded as follows:

STATUS_UPDATE: StatusUpdate= <Status_Update> <newline>

“STATUS_UPDATE” MUST be capitalized, and MUST immediately follow a newline character. The fields MUST be separated by a single space character and the protocol MUST ignore any unknown fields.

<STATUS_UPDATE> is an ASCII encoding and MUST be one of the characters a, b, c, d, e, f, g, h, i, j, k, l.

ABNF Grammar
==========

connection = 1*(msg newline)
msg = maze_update | pacman_arrived | foreign_pacman_left | pacman_died | pacman_go_home | pacman_update | ghost_update | ghost_was_eaten | eat | score_update| status_update

maze_update = “MAZE_UPDATE:" space maze_data 1* (space maze_update_field)

maze_data = 812(maze_update_field)

pacman_arrived= “PACMAN_ARRIVED:" space pacman_arrived_field 1* (space pacman_arrived _field)

pacman_arrived_field= “XPos=” integer | “YPos=” integer | “Direction=” integer

foreign_pacman_left=“FOREIGN_PACMAN_LEFT"

pacman_died= “PACMAN_DIED"

pacman_go_home= “PACMAN_GO_HOME"

pacman_update= “PACMAN_UPDATE:" space pacman_update_field 1* (space pacman_update_field)

pacman_update_field= “XPos=” integer | “YPos=” integer | “Direction=” integer

ghost_update= “GHOST_UPDATE:" space ghost_update_field 4(space ghost_update_field)

ghost_update_field= “GhostNum=” integer | “XPos=” integer | “YPos=” integer

ghost_was_eaten= “GHOST_WAS_EATEN:" space ghost_was_eaten_field 1* (space ghost_was_eaten _field)

ghost_was_eaten_field= “GhostNum=” integer | “Local=” integer

eat= “EAT:" space eat_field 1* (space eat_field)

eat_field= “XPos=” integer | “YPos=” integer | “Local=” integer

score_update= “SCORE_UPDATE:” space score_update_field 1* (space score_update_field)

score_update_field= “Score=” integer

status_update= “STATUS_UPDATE:” space status_update_field 1* (space status_update_field)

status_update_field= “StatusUpdate=” integer

ABNF Grammar

==========

integer = POS-DIGIT *(DIGIT)

alpha-numeric = ALPHA | DIGIT

DIGIT = "0" | POS-DIGIT

POS-DIGIT = "1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

space = %d32

newline = %d10

CHAR = %x01-7F