Distributed Mazewar Specification
Protocol Version 0

ALI YAHYA MATT SPARKS

April 14–27, 2010

1 Overview

This specification defines the rules of the Mazewar game as well as the protocol for gameplay over a network. The protocol defined in this document is the first version (version 0).

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC 2119[1].

2 Description of Mazewar

Mazewar is a distributed, multiplayer game in which each player is a rat that moves around in a maze and tags opposing players. When a player encounters his opponents, he can tag them by firing a projectile. A player receives points for tagging other rats and loses points for being tagged.

2.1 World

The maze consists of a 32×16 array of cells. An example maze is shown in Figure 1. Each cell is occupied by a player, occupied by a wall, or is empty.

2.2 Vision

Throughout the game, each player has two view panels into the state of the game. The first is the perspective window, which shows the player's first person perspective as he or she navigates the maze. In the perspective window, a player sees any opposing players in line of sight. The second is a top level view of the maze showing the player's

[1] RFC 2119: http://www.ietf.org/rfc/rfc2119.txt

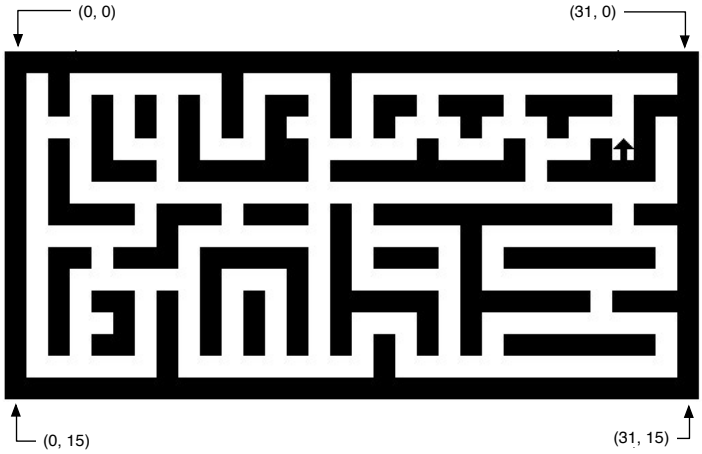

Figure 1: Example maze.

position and orientation from above. If the player has fired a projectile, the top level view reveals its position on the board as it travels. The top level view does not reveal the position of any other player in the game, but may show the positions of their fired projectiles.

2.3 Movement

A player can move forward or backward, turn left or right, or peek down hallways without exposing him/herself. However, no more than a single player can occupy a square a time.

2.4 Projectiles

Players can fire projectiles at their opponents. Projectiles always travel in a straight line and at a constant speed of 200 milliseconds per square. Projectiles travel in the direction in which the player who launched the projectile was facing at the time of launch. At any given time, a player has exactly zero or exactly one projectile in flight. A projectile can travel through other projectiles or the player that fired it unimpeded and without causing any harm.

Projectile movement is discretized. A projectile fired at $t = 0$ is considered to remain in the square from which it was fired while $t < 200$ milliseconds. When $t \geq 200$ ms and $t < 400$ ms, the projectile is considered to be in the adjacent square, and so on. A projectile *expires* when it hits a wall, i.e., moves into a wall square, or tags another player. In the case of tagging another player, the projectile must not expire until the tagged player notifies the player who launched the projectile. Once the projectile has expired, the player may immediately fire another projectile.

2.5 Tagging

A player that is touched by an opponent's projectile is tagged. More formally, a rat that exists in a square at any time during which a projectile simultaneously occupies that square is considered tagged. A tagged player loses 5 points and a player that tags another player gains 11 points. The cost of firing a projectile is 1 point. Whenever a player is tagged, he/she respawns at a random, available position in the maze with a random orientation (though not facing a wall).

3 Network Play

This section describes how local state is disseminated to other players over a network and how local state is updated by network messages from other players.

3.1 Joining a Network Game

A Mazewar game is joined by subscribing to a well-known IP multicast group and listening on a well-known port. No messages need to be sent to the group when joining; periodic messages from other players will be sufficient to learn their states within a short period of time.

When joining a network game, the implementation must relocate the local player to a random unoccupied square in the maze.

3.2 Message Structure

Once subscribed to an IP multicast group, local state must be sent to other players subscribed to the same IP multicast group on the same port. Each message sent to the multicast group by a player has a common packet structure. The packet format is shown in Figure 2. The message must be at least 28 bytes in size and at most 36 bytes. Messages of other sizes must be discarded. Each field is represented in network order (big endian).

Version (bits 0–2)

Implementations of this document should zero these bits to indicate protocol version 0. Implementations of this specification must ignore messages with non-zero values in this field.

Message Type (bits 3–5)

This field has a value in the range 0 to 3 inclusive, and identifies the message type. All possible message types and their semantics are explained in Section 3.3.

Projectile Present (bit 6)

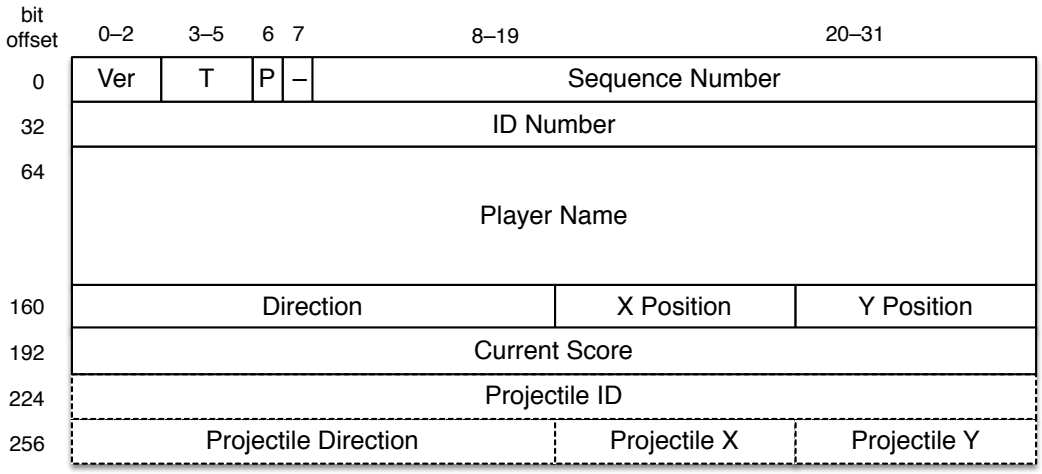

Figure 2: Network packet format.

If this bit is set, the Projectile ID, Projectile Direction, Projectile X Position, and Projectile Y Position fields must be present and valid in the message. If this bit is not set, these fields may be excluded from the message and must be ignored if they are present.

Unused (bit 7)

This bit is unused and must be ignored. An implementation may set this bit to zero.

Sequence number (bits 8–31)

The sequence number field is used to indicate the order of messages sent from a node. The sequence number strictly increases: each new message must contain a sequence number higher than in the previous message. Implementations must ignore messages from a remote node with sequence numbers less than or equal to sequence numbers already seen from that node. Sequence number state must only be reset in the case of a timeout as described in Section 3.6.

ID Number (bits 32-63)

This field must contain a random 32-bit value chosen by the implementation each time a network game is entered. This value is to be used in combination with the source IP address and source port of network messages to identify a unique player.

Player Name (bits 64–159)

Human-readable name of the local player. This field should be formatted with a printable ASCII string. Printable characters are defined as decimal ASCII values between 32 and 126, inclusive. This field must be between 1 and 12 characters, inclusive. The first byte in this string is in bits 32–39 of the message structure. Unused bytes at the end of the string must be set to null character (`'\0'`). Unprintable characters should be replaced with printable characters by the implementation.

Direction (bits 160–175)

This field indicates the direction a rat is facing. Four values are valid for this field: 0 (North), 1 (South), 2 (East), 3 (West). Messages with values not in this range must be ignored. The top of the maze is West, bottom is East, left is South, and the right side is North.

X Position (bits 176–183)

The X position of the rat is stored in this field. Values 0 to 31, inclusive, are valid. Messages with values larger than 31 must be ignored.

Y Position (bits 184–191)

The Y position of the rat is stored in this field. Values 0 to 15, inclusive, are valid. Messages with values larger than 15 must be ignored.

Current Score (bits 192–223)

The current score of a player is stored in this field. All signed 32-bit values are valid in this field. A score can be negative or positive.

Projectile ID (bits 224–255)

This field is used to uniquely identify a projectile fired by a player. This field may be optional depending on the message type.

Projectile Direction (bits 256–271)

The projectile direction is the direction in which the projectile is traveling. This field has the same semantics as the Direction field. This field may be optional depending on the message type.

Projectile X Position (bits 272–279)

The X position of the projectile is stored in this field. This field has the same semantics as the X Position field. This field may be optional depending on the message type.

Projectile Y Position (bits 280-287)

The Y position of the projectile is stored in this field. This field has the same semantics as the Y Position field. This field may be optional depending on the message type.

3.3 Message Types

There are five different message types that all implementations must support. All five have the structure described in Section 3.2 but have different semantics. The type of a message is indicated by the 3-bit Message Type field in the network packet.

The sections below explain the five message types and their semantics. The requirements for when these messages are to be sent are also explained within these sections. The effects on local state when receiving these messages are described in Section 3.6.

3.3.1 STATEUPDATE Message (type 0)

All clients must periodically multicast STATEUPDATE network messages to inform all other game participants of their local state. It is recommended that the time interval between updates is between 50 and 100 milliseconds. The interval must not be more than 300 milliseconds.

Metadata The Version, Message Type, Sequence Number, ID number, and Player Name fields must be populated as explained in Section 3.2.

Current Position The Direction, X Position, Y Position fields must be populated with the direction and position of the local player's rat. The descriptions and constraints of these fields are noted in Section 3.2. Each (x, y) coordinate represents a square in the maze, where $(0, 0)$ is the upper left square and $(31, 15)$ is the lower right square as shown in Figure 1.

Current Score The local player's score must populate the Current Score field.

Projectile Information If a client has a projectile in flight at the time a STATEUPDATE message is sent, the Projectile Present bit must be set, and the projectile information fields must be populated as described in Section 3.3.2. If a projectile is not in flight, the Projectile Present bit must be left unset and the projectile information fields may be omitted.

3.3.2 FIRE Message (type 1)

Whenever a projectile is fired by the local player, the client must immediately multicast a FIRE network message to all other game participants. The metadata, current position, and current score fields must be populated in the FIRE message identically to how these fields are populated in a STATEUPDATE message as described in Section 3.3.1.

Projectile Information The Projectile Present bit must be set and the Projectile ID, Projectile Direction, Projectile X Position, and Projectile Y Position fields should be populated with the current location of the projectile. Because FIRE messages are generated immediately after firing a projectile, these fields must be equivalent to the Direction, X Position, and Y Position fields, respectively, which represent the position of the local player's rat.

3.3.3 TAGGED Message (type 2)

Whenever the local player is tagged by an opponent's projectile, the client must immediately multicast a TAGGED network message to all other game participants. The metadata fields must be populated as described in Section 3.3.1.

Current Position As described in Section 2.5, being tagged causes rat to respawn to a new location in the maze. In a TAGGED message, the position fields should reflect the new position after the respawn. In the case of resending a TAGGED message, the position fields must reflect the most current position, which may not be the position moved to as the result of the last respawn.

Current Score The Current Score field must be populated with the new score after the necessary tagging adjustment is made (Section 2.5).

Projectile Information The Projectile Present bit should be set and the Projectile ID field must be populated with the ID of the projectile that tagged the local player. The projectile direction and position fields must be populated with the direction and position of the projectile at the time of the tagging.

A TAGGED message for a projectile must be resent periodically until a TAGGEDACK message is received for it. The interval between retransmissions of a TAGGED message should be between 1 and 5 times the interval for STATEUPDATE messages.

If the local player is tagged by, say, two projectiles before the first TAGGED message is acknowledged with a TAGGEDACK, he should begin sending two TAGGED message at the next retransmission interval.

3.3.4 TAGGEDACK Message (type 3)

This message type applies only after the local player has fired a projectile. Thereafter, every time a TAGGED message is received from an opponent, the local player must send a TAGGEDACK message in response.

Current Score The Current Score field must be populated with the new score after the necessary tagging adjustment is made (Section 2.5).

Projectile Information All projectile related fields, including the Projectile Present bit, must be set to values that are equivalent to those in the TAGGED packet that is being acknowledged.

3.3.5 QUIT Message (type 4)

Whenever the local player voluntarily quits the game, the client should multicast a QUIT network message to all other game participants. The metadata fields must be populated as described in Section 3.3.1. The message must be of a valid length, but all other fields (position, score, and projectile information) may or may not be populated.

3.4 Local State in Network Play

To facilitate network play, some local state must be kept to support communication over an unreliable IP multicast channel. This section describes the implementation requirements for sequence numbers, projectile IDs, and how to identify other players reasonably uniquely.

3.4.1 Sequence Numbers and Projectile IDs

When joining a game, a player must start with a sequence number of 1. When a player A sends a new message to the multicast group, the sequence number in the must be greater than all other sequence numbers used by A in A's current session. If A leaves and rejoins, A may reset its sequence number to 1.

Projectile IDs must be decided more carefully. A player sending a TAGGED message indicates the ID of the projectile by which he was tagged, but projectile IDs are not strictly unique across all players. To make projectile IDs reasonably unique, an implementation must choose the initial projectile ID by setting the upper 16 bits of the 32-bit field to a random value between 0 and 65,535 inclusive, and the lower 16 bits to 0x0001. This must be done exactly once at the start of each session. Subsequent projectile IDs have the same semantics as sequence numbers; each new projectile must have an ID larger than all other projectile IDs used by the same player in the session.

3.4.2 State of Other Players

An implementation must be aware of the most recently used sequence numbers and projectile IDs of other players until this state is no longer required as a result of a timeout or a QUIT. The IP address and port number of an incoming message is sufficient to uniquely identify a player. That is, recent sequence numbers and projectile IDs should be kept for each unique *<IP address, Port, ID Number>* player tuple. The position, direction, score, and projectile information for other players need also be kept for the graphical display.

3.5 Dead Reckoning for Projectiles

When a FIRE or STATEUPDATE message is received that indicates the presence of a new projectile, an implementation must keep track of the starting position and direction for that projectile ID, as well as the time at which it was observed to be fired. The positions of that projectile can then be *dead reckoned* using the known parameters of the projectile. That is, the projectile must move in local state without any further updates to that projectile ID.

Subsequent updates of that same projectile ID through new STATEUPDATE messages should be used to correct the position of the projectile. However, updates to the projectile's position should only be made if the projectile will move forward with the update applied.

3.6 Receiving Messages

The state diagram shown in Figure 3 is a representation of the local client's view of a single remote opponent (herein referred to as opponent X). As described in Section 3.3, there are five types of messages that the local client can possibly receive from X. As different messages are received, the local client transitions between different states and reproduces any side effects in the player's view of the game. Figure 3 shows the different states the client can be in. The arrows represent transitions between states, and the rectangular labels indicate the causes of those transitions.

This section describes the possible transitions that can occur as a result of receiving messages from a single opponent X, as well as the visible side effects that those transitions must produce in the local game state. Implementations of this protocol should generalize the workflow described by maintaining a state machine like the one in Figure 3 for each opponent.

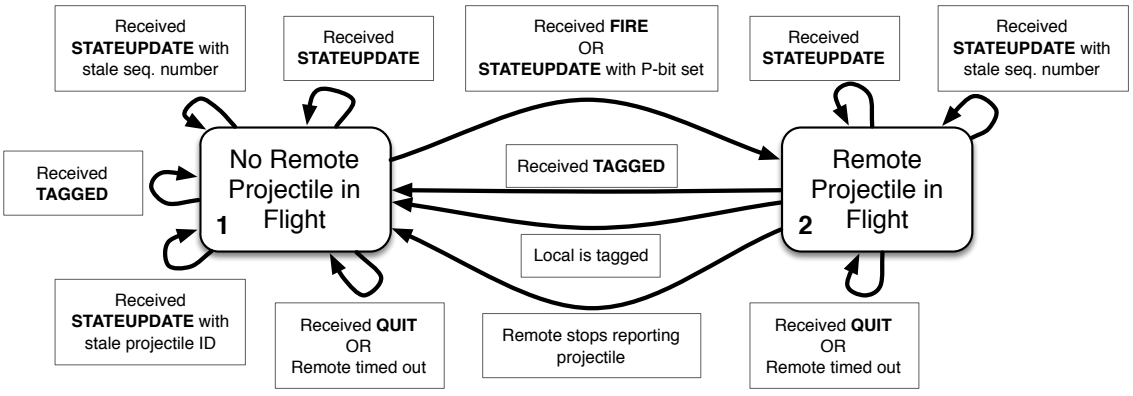

Figure 3: Receiving Messages State Diagram

Initially, before opponent X fires any projectiles, the local client is in the state that is labeled "No Remote Projectile in Flight." Consequently, that is the starting state of the diagram.

The title of each sub-section below corresponds to an event (displayed as a rectangular label) in Figure 3.

3.6.1 Received STATEUPDATE

Cause This event occurs when the local client receives a STATEUPDATE message from opponent X. As described in Section 3.3.1, clients must periodically send STATEUPDATE messages to all other game participants.

Side effects Upon receiving a message of this kind, the local client must update the position, direction, and score of the opponent X on the local representation of the board.

If, at that time, the local player has the opponent X's updated position in view on the perspective window, then opponent X must be made visible.

If, after updating X's position in local state, both the local player and X occupy the same square in the maze, the implementation must relocate the local player to an unoccupied square on the maze *if and only if* he has a sequence number less than or equal to that of X.

Also, if the STATEUPDATE message has the projectile bit (Section 3.2) set, the projectile's position must be updated in the local representation of the board. However, projectiles should never visibly move backward. If the reported position of the projectile is behind its local dead-reckoned (Section 3.5) position, then it should be ignored.

3.6.2 Received STATEUPDATE with stale seq. number

A *stale sequence number* is defined as a sequence number that is less than or equal to the highest sequence number observed for the remote user.

Cause This event can occur as a result of the network's inherent unreliability. For instance, any given packet from opponent X can be delayed and arrive after subsequent packets from opponent X.

Side effects STATEUPDATE messages with stale sequence numbers must be ignored by the local client.

3.6.3 Received STATEUPDATE with stale projectile ID

A *stale projectile ID* is defined as a projectile ID that is less than or equal to the highest projectile ID from opponent X that has expired in local state.

Cause This event can similarly occur as a result of network unreliability. However, this event can also occur if opponent X doesn't receive the tagged opponent's TAGGED message (Section 3.6.6) in time, and sends another STATEUPDATE update about the projectile.

Side effects The local client must treat the message as if its *projectile present* bit were set to zero (refer to Section 3.2).

3.6.4 Received FIRE, OR STATEUPDATE with P-bit set

This event causes a transition from state 1 to state 2. This subsection is a description of that transition. For a description of the transition that results from receiving a STATEUPDATE message while already in state 2, refer to Section 3.6.1.

Cause This event can occur in two different cases:

1. A FIRE message is received from opponent X because he/she has fired a projectile. Note that, because players can have at most one projectile in flight at a time, FIRE messages from opponent X can only be received when the local client is in state 1.

2. A STATEUPDATE message is received from opponent X with the projectile bit set. Such a message may be received before FIRE if the FIRE message was lost in the network or is delayed.

Side effects Upon detecting this kind of event, the local client must create the fired projectile in its the local representation of the board. Regardless of the projectile's position, an implementation may also choose to make it visible in the top-view perspective.

If opponent X has an active projectile in local state with a projectile ID older than the projectile ID in the received message, the existing projectile should expire immediately and be replaced by the projectile described in the received message.

The implementation must also process the position, direction, and score as described in Section 3.6.1.

3.6.5 Received QUIT, OR remote timed out

Cause This event can occur in two different cases:

1. A QUIT message is received from opponent X because he/she left the game voluntarily.
2. No STATEUPDATE message, *with a new sequence number*, is received from opponent X in 3000 ms or more. Note that STATEUPDATE messages with old sequence numbers do not reset the timer.

Side effects Upon receiving a message of this kind, regardless of its cause, the local client must remove opponent X from the visible representation of the board and delete all local state associated with opponent X.

3.6.6 Received TAGGED

Regardless of whether the local client is in state 2 or state 1, the end state of this event is state 1.

Cause This event occurs when the local client receives a TAGGED message from someone other than opponent X indicating that he/she was tagged by a projectile fired by opponent X.

Side effects Upon receiving this kind of message, the local client must remove both the projectile and the tagged player from the visible representation of the board.

3.6.7 Local is tagged

This event causes a transition from state 2 to state 1.

Cause This event occurs when the local client is the player that was tagged by opponent X's projectile in flight.

Side effects Upon detecting this kind of event, the local client must remove both its player and the projectile from the visible representation of the board and must respawn. For information about re-spawning, refer to Section 2.5.

3.6.8 Remote stops reporting projectile

This event causes a transition from state 2 to state 1.

Cause This event occurs when opponent X's periodic STATEUPDATE messages cease to have the projectile bit (Section 3.2) set. This can happen if the projectile simply misses all other oponents and hits a wall.

Side effects Upon detecting this kind of event, the local client must remove the projectile from the visible representation of the board.