CS244b Assignment 1:

Mazewar Protocol Specification

Hrishikesh Gadre <hgadre@stanford.edu>
Alhaad Gokhale <alhaad@stanfrod.edu>

April 10, 2016

Introduction

The Mazewar game and its requirements are described at
http://www.stanford.edu/class/cs244b/mazewar.html.
This document describes a protocol to achieve those requirements with an aim of responsive local UI and eventual consistency of the global game state.

Constants

MULTICAST_INTERVAL - 250 milliseconds
PROJECTILE_SPEED - 1 cell / 250 milliseconds
CLOAKING_TIME - 5 seconds
CLOAKING_RECUPERATION_TIME - 10 seconds

Protocol Definition (Version 1)

The Mazewar protocol defines a way in which the players communicate with each other to synchronize the game state. It consists of a set of descriptors used for communicating data packets between players and a set of rules governing the inter-player exchange of descriptors.

Every Mazewar game uses a dedicated UDP multicast group for communication. Hence for a player to participate in the game, he/she must know the relevant information about this multicast channel i.e. a UDP port which is a part of a multicast group.

Following descriptors / packet types are defined,

Descriptor	Descriptor ID	Description
State	1	Communicates comprehensive information about the

		player state which includes (a) position and direction of rat (b) position and direction of the projectile and (c) the absolute value of the player score
TagRequest	2	Communicates the possibility that local rat may be tagged by the projectile sent by the remote rat. For a given projectile, one or more players may send this descriptor. It is the responsibility of the projectile sender to figure out the appropriate target player (e.g. based on the distance between source and destination) and send the explicit acknowledgment in the form of Tagged descriptor.
TagResponse	4	Communicates the fact that a given rat is hit by a projectile sent by this rat. This is the confirmation descriptor for one or more TagRequest descriptors sent earlier.

Descriptor format

Descriptor header

Offset	Field	Length(bits)	Default value	Purpose
0	version	8	x0001	Specify the version of the protocol used by the player. If message is received with a different version number, a player should drop it silently.
8	descriptor_type	8		Specify the type of the descriptor
16	payload_length	16		Length of the descriptor payload
32	player_id	32		Unique identifier of the player which is sending this descriptor. It is the responsibility of the implementation to make sure that the player_id is unique and handle the case where two or more players use the same player_id.

64	sequence number	32		Monotonically increasing sequence number (only during a single player session) used to uniquely identify the descriptor.

Immediately following the descriptor header, is a payload consisting of one of the following descriptors:

State

The *State* descriptor communicates comprehensive information about the player state which includes (a) position and direction of rat (b) position and direction of the projectile and (c) the absolute value of the player score. Every player is responsible to send this descriptor periodically. This descriptor *could also* be used as a heartbeat mechanism to detect the *liveness* of the player.

This specification makes no recommendations as to the frequency at which a player should send *State* descriptors, although player implementers should make every attempt to minimize *State* traffic on the network.

For the State descriptor, the descriptor_type field in the header is 1. The State descriptor payload contains following information,

Offset (bits)	Field	Length (bits)	Purpose
0	rat_dir	4	The direction of the rat
4	projectile_dir	4	The direction of the projectile
8	collision_token	24	A 24 bit token used to resolve the conflicts where two or more player attempt to move to the same cell. A player with the smallest *collision_token* value will be allowed to move to this cell. The value is randomly generated if not set to 0 as described in the semantics.
32	rat_x_pos	16	The X coordinate of the rat
48	rat_y_pos	16	The Y coordinate of the rat
64	projectile_seq	32	The monotonically increasing sequence number of the projectile. This helps to assign a unique id for every projectile sent by a

			given player. This is defined to ensure that multiple players have common understanding for which projectile they are communicating about. This helps in resolving the problems due to delayed packets. When no active projectile in progress, the value of this field should be 0xffffffff
96	projectile_x_pos	16	The X coordinate of the projectile. When no active projectile in progress, the value of this field should be 0xffff
112	projectile_y_pos	16	The Y coordinate of the projectile When no active projectile in progress, the value of this field should be 0xffff
128	score	32	The absolute value of the player score.
160	player_name	256	The display name chosen by the player.

The direction of the rat (and the projectile) is encoded as follows,

Direction	Encoding
Not applicable (e.g. when no projectile is active or if the rat is cloaked)	0000
North	0001
South	0010
East	0100
West	1000

TagRequest

This descriptor communicates the possibility that local rat may be tagged by the projectile sent by a remote rat. For a given projectile, one or more players may send this descriptor. It is the responsibility of the projectile sender to figure out the appropriate target player (e.g. based on the distance between source and destination) and send the explicit acknowledgment in the form of TagResponse packet.

For the *TagRequest* descriptor, the descriptor_type field in the header is 2. The *TagRequest* descriptor payload contains following information,

Offset (bits)	Field	Length (bits)	Purpose
0	shooter_id	32	Player id of the shooter
32	projectile_seq	32	The sequence number of projectile under consideration.
64	rat_x_pos	16	The X coordinate of the rat
80	rat_y_pos	16	The Y coordinate of the rat
96	rat_dir	4	The direction of the rat (or 0000 if the rat is currently cloaking).
100		28	unused

Please note the information about the rat position provided in the payload corresponds to the player identified by the “player_id” field in the header.

TagResponse

This descriptor communicates the fact that a given rat is hit by a projectile sent by this rat. This is the confirmation descriptor for one or more TagRequest descriptors sent earlier.

For the *TagResponse* descriptor, the descriptor_type field in the header is 2. The *TagResponse* descriptor payload contains following information,

Offset (bits)	Field	Length (bits)	Purpose
0	projectile_seq	32	The sequence number of projectile under consideration.
32	player_id	32	Identifies the player which is tagged by this projectile.

Protocol Semantics / Timing

Joining the game

A client that wants to join the game must listen to all the packets on the multicast group for at least 4 * MULTICAST_INTERVAL and build the state of the game before sending the first *State* packet to the UDP multicast group.

If the client is the first player in the multicast group (i.e. it does not see any traffic), it still needs to send the State packet.

Leaving the game

A client which wants to leave a game can do so any time by simply not sending any more packets.

Rat move

A client updates the position and / or direction of its rat by sending a *State* packet with the updated position and / or direction.

Rat collision resolution

Rat collision happens if two rats try to move into the same cell at the same time. Since, two rats are not allowed in the same cell at the same time, resolution is done by randomly allowing only one rat to enter that cell. It is expected that when two (or more) rats collide, we would prefer a visible rat to enter the cell.

If a client observes that the *State* packet of another rat says that it is in the same position (irrespective of direction) as itself, it compares the random collision_token and moves to its previous position (by sending another *State* packet with collision_token set to 0).

Rat cloaking

Rat cloaking is done by setting the direction to 0000 in the *State* packet. The client must report the current rat_x_pos and rat_y_pos values even when the rat is cloaking (since these values are necessary to detect the rat collisions).

The client is expected to make sure that cloaking only lasts for CLOAKING_TIME and new clocking cannot begin until after CLOAKING_RECUPERATION_TIME.
A client which is painting another rat on screen must make sure that the other rat is not displayed if its direction is set to 0000.

Projectile firing / moving

The projectile information is added by a client into its *State* packet and updated as the projectile moves at the rate of PROJECTILE_SPEED. The client removes projectile information from the *State* packet if the projectile either hits a wall or tags another rat. Tagging another rat is described later in this document.

Rat getting tagged (i.e. hit by a missile)

If a client finds that a projectile is in the same cell as itself, it must a *TagRequest* packet (which contains the projectile sequence number, and shooter's player id) repeatedly (every MULTICAST_INTERVAL) until it receives a *TagResponse* (with the appropriate missile seq number and player id of the rat that got tagged).
If the seq number and player id match, the rat must decrease its score by 5 points (7 points if it was cloaked when hit).

Projectile hits another rat

A client knows that its projectile has hit another rat if it receives a *TagRequest* message with its player id as the shooter id. It must respond to every *TagRequest* with an appropriate *TagResponse*.

The client also increases its score by 11 points (13 if the other rat was cloaked). The client can know if the other rat was cloaked by looking at the direction in the *TagRequest* packet.

It is possible (though unlikely) that two different rats send *TagRequest* packets, and it is the responsibility of the shooter to decide which rat got tagged based on information in its state and in the *TagRequest* packet. This means that for a projectile seq number used by the client, all *TagResponse* packets concerning that projectile should have only one player_id.

Loss of contact

A client which has not received a *State* packet from another rat for more than 4 * MULTICAST_INTERVAL can remove that rat from its local game state.