GameState

INHERITS FROM Object

DECLARED IN OpponentApp/GameState.h

CLASS DESCRIPTION

The GameState class maintains the state of a Ragnarok game: where the pieces are, what turn it is, and the game history. When a Ragnarok application communicates with another Ragnarok application over the network, or with a computer opponent, it sends instances of the GameState class.

CONSTANTS AND DEFINED TYPES

The 37 pieces are Ragnarok are numbered as follows: 0⋯23 are the White pawns, 24⋯35 are the Black pawns, and 36 is Loki.

The constants **CENTER**, **CORNER, OFFBOARD**, and **PLAIN** refer to whether a particular location is the center of the board, a corner, an off-board (unused) square, or just a plain old square.

The constants **NOBODY, W_PAWN**, **B_PAWN**, and **LOKI** refer to the occupancy of a square.

The constants **BLACK**, **WHITE**, **BLACK_WON**, **WHITE_WON**, and **DRAW** refer to the state of the game (how the game ended, or whose turn it is).

In Ragnarok, squares are labeled a⋯k horizontally, and 1⋯11 vertically. Inside the Ragnarok program, they are referenced by as ordered pairs: <0⋯10, 0⋯10>. For efficiency, in the GameState data structures, locations

are encoded as unsigned char's. The macro **XYTONUM**(*x*,*y*) returns the encoding of the ordered pair <*x*,*y*>. The macro **NUMTOX**(*num*) returns the first element of the pair encoded by *num*, and **NUMTOY**(*num*) returns the second element of the pair encoded by *num*. Adding **EAST** to an encoding *num* results in the encoding of the location one square east of the location referenced by *num*. **WEST**, **NORTH**, and **SOUTH** work similarly.

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in GameState	unsigned char	pieceLocs[37];
	struct spot {	
	unsigned char	who;
	unsigned char	idnum;
	} pieces[256];	

```
unsigned char       whoseTurn;
unsigned char       numPawns[2];
struct move {
    unsigned char       from;
    unsigned char       to;
}   moves[1024];
short               numMoves;
struct capture {
    short               when;
    unsigned char       where;
    unsigned char       idnum;
}   captures;
short numCaptures;
```

pieceLocs — The encoded locations of the pieces.

pieces	A list of the piece types and locations, indexed by location on the board.
whoseTurn	The state of the game: White's turn, Black's turn, game drawn, White victory, or Black victory.
numPawns	How many pawns each side has.
moves	The moves made in the game.
numMoves	How many moves have been made in the game.
captures	The captures made in the game.

numCaptures	How many captures have been made in the game.

METHOD TYPES

Initializing the class	+ initialize
Initializing a new GameState	- init - resetState
Making moves	- makeMove: - makeWhiteMove: - makeBlackMove:
Undoing moves	- undoMove - undoWhiteMove

	- undoBlackMove
Questions about moves	- anyLegalMoves - checkMove:
Archiving	- read: - write:

CLASS METHODS

initialize

+ **initialize**

Prepares internal class variables. Returns **self**.

INSTANCE METHODS

anyLegalMoves

- (BOOL)**anyLegalMoves**

Returns YES if there are any legal moves from the current position.

checkMove

- (BOOL)**checkMove:**(struct move)*request*

Returns YES if *request* is a legal move from the current position.

init

- **init**

Initializes the GameState, which must be a newly allocated GameState instance. Returns **self.**

makeMove:

- (void)**makeMove:**(struct move)*request*

Makes the move *request*, which should be a legal move. The legality of the move is not checked, so be careful. This method simply calls **makeWhiteMove:** or **makeBlackMove:**, depending on whose turn it is.

makeWhiteMove:

- (void)**makeWhiteMove:**(struct move)*request*

Makes the move *request*, which should be a legal move for White to make (i.e. the move is legal and it's White's turn). The legality of the move is not checked, so be careful.

makeBlackMove:

- (void)**makeBlackMove:**(struct move)*request*

Makes the move *request*, which should be a legal move for Black to make (i.e. the move is legal and it's Black's turn). The legality of the move is not checked, so be careful.

read:

- **read:**(NXTypedStream *)*stream*

Reads the GameState from the typed stream *stream*.

resetState

- (void)**resetState**

Resets the GameState to the starting position.

undoMove

- (void)**undoMove**

Undoes the last move made in the GameState, in which there should be at least one move made. This method simply calls **undoWhiteMove** or **undoBlackMove**, depending on whose turn it was.

undoWhiteMove

- (void)**undoWhiteMove**

Undoes the last move made in the GameState, in which there should be at least one move made. Also, it should be Black's turn (so that the last move made was a White move).

undoBlackMove

- (void)**undoBlackMove**

Undoes the last move made in the GameState, in which there should be at least one move made. Also, it should be White's turn (so that the last move

made was a Black move).

write:

- **write:**(NXTypedStream *)*stream*

  Writes the GameState to the typed stream *stream*.