Project Multitris

Group 23
Marcus Dicander
Måns Olson
Tomas Alaeus
Daniel Boström
Oscar Olsson

Section 5.5

The methods and fields are in the javadoc documentation format, ordered by class according to the index below:

Table of Contents

Interface ApplicationState

All Known Implementing Classes:
GameSessionState, LobbyState, MenuState

```
public interface ApplicationState
```

This is the interface for the different application states.

Method Summary

void	**doInputAction**(java.lang.String action) Updates the state according to user action.
void	**render**(Graphics g) Renders the current state.

Method Detail

render

```
void render(Graphicsg)
```

Renders the current state.

Parameters:
`g` - The graphics context upon which to render the state.

---

doInputAction

```
void doInputAction(java.lang.Stringaction)
```

Updates the state according to user action.

Parameters:
`action` - A String representing an action.

Class ApplicationStateManager

```
java.lang.Object
  └ApplicationStateManager
```

---

public class **ApplicationStateManager**extends java.lang.Object

Keeps track of the current states and switches between them. Input provided by the player is routed to the state with the lowest int index. Network actions are routed to the GAME if possible, otherwise to the LOBBY state. This class will extend the BasicGame class and thus have an update loop. This loop will be responsible for gathering network commands, and will call forwardNetworkActions.

Field Summary

private boolean[]	**activeStates** The states currently active.
static int	**GAME** Gamestate.
static int	**LOBBY** Gamestate.
static int	**MENU** Gamestate.
static ClientCommunication	**network** For server communication.
private ApplicationState[]	**states** Contains the actual states.

Constructor Summary

ApplicationStateManager()	

Method Summary

void	**ApplicationStateManager**() Constructor for the ApplicationStateManager class
void	**forwardInput**(java.lang.String action) Forwards actions from InputManager to the current state.
void	**forwardNetworkActions**(java.lang.String[] actions) Forwards server actions from the ClientCommunication class to current state.
static void	**main**(java.lang.String[] args) This is the main method called on application startup.
void	**setStateActive**(int state, boolean value) Sets a state as active.
void	**switchState**(int state) Change the current state.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

MENU

```
public static final int MENU
```

Gamestate.

See Also:
Constant Field Values

LOBBY

```
public static final int LOBBY
```

Gamestate.

See Also:
Constant Field Values

GAME

```
public static final int GAME
```

Gamestate.

See Also:
Constant Field Values

---

network

```
public static ClientCommunication network
```

For server communication.

---

states

```
private ApplicationState[] states
```

Contains the actual states.

---

activeStates

```
private boolean[] activeStates
```

The states currently active.

Constructor Detail

ApplicationStateManager

`public` **`ApplicationStateManager`**`()`

Method Detail

ApplicationStateManager

`public void` **`ApplicationStateManager`**`()`

Constructor for the ApplicationStateManager class

---

switchState

`public void` **`switchState`**`(intstate)`

Change the current state.

Parameters:

`state` - One of the three predefined states MENU, LOBBY and GAME.

---

forwardInput

`public void` **`forwardInput`**`(java.lang.Stringaction)`

Forwards actions from InputManager to the current state.

Parameters:

`action` - A String representing an action.

---

forwardNetworkActions

`public void` **`forwardNetworkActions`**`(java.lang.String[]actions)`

Forwards server actions from the ClientCommunication class to current state. This method is called by the program's update loop.

Parameters:

`actions` - An array of Strings representing actions to perform.

---

setStateActive

```
public void setStateActive(intstate,
                           booleanvalue)
```

Sets a state as active.

Parameters:

`state` - The state to be set.
`value` - true for active.

---

main

```
public static void main(java.lang.String[]args)
```

This is the main method called on application startup.

Class Board

```
java.lang.Object
  └Board
```

```
public class Boardextends java.lang.Object
```

A class representing a Board.

Field Summary

Brick [][]	**board** A matrix containing the Bricks in this Board.

Constructor Summary

Board(int width, int height) Constructor for Board.	

Method Summary

void	**fixPiece**(Piece piece) Fixates a Piece on the Board by splitting it into Bricks and moving them to this Board.
Brick	**getBrick**(int x, int y) Gets the Brick in a specific position.
Brick	**removeBrick**(int x, int y) Removes the Brick at the specified position.
int[]	**removeRow**(int y) Removes all the Bricks in the specified row.

Brick	**setBrick**(int x, int y, Brick brick) Adds a Brick to the specified position.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

board

```
public Brick[][] board
```

A matrix containing the Bricks in this Board.

Constructor Detail

Board

```
public Board(intwidth,
             intheight)
```

Constructor for Board.

Parameters:

`width` - The width of the Board.
`height` - The height of the Board.

Method Detail

removeRow

```
public int[] removeRow(inty)
```

Removes all the Bricks in the specified row. All the bricks above the specified row that has been fixated will move down one step. If any of the removed bricks contains a powerup it will be extracted and returned. This method is called by doActions in the GameLogic class.

Parameters:

`y` - The index of the row to be removed.

Returns:

A list containing the extracted powerups.

See Also:

GameLogic

---

fixPiece

```
public void fixPiece(Piecepiece)
```

Fixates a Piece on the Board by splitting it into Bricks and moving them to this Board. This method is called by doActions in the GameLogic class.

Parameters:

`piece` - The Piece to fixate.

See Also:

GameLogic

---

removeBrick

```
public Brick removeBrick(intx,
                         inty)
```

Removes the Brick at the specified position. This method is called by doActions in the GameLogic class.

Parameters:

`x` - The x-coordinate of the Brick to be removed.

`y` - The y-coordinate of the Brick to be removed.

Returns:

The Brick that was removed.

See Also:

GameLogic

---

setBrick

```
public Brick setBrick(intx,
                      inty,
                      Brickbrick)
```

Adds a Brick to the specified position. If the specified position already contains a brick it will be returned. This method is called by doActions in the GameLogic class.

Parameters:

`x` - The x-coordinate of the brick to be added to the Board.

`y` - The y-coordinate of the brick to be added to the Board.

`brick` - The Brick to be added to the Board.

Returns:

The previous Brick the specified position, or null if the specified position was empty.

See Also:

GameLogic

---

getBrick

```
public Brick getBrick(intx,
                      inty)
```

Gets the Brick in a specific position. This method is called by doActions in the GameLogic class.

Parameters:

`x` - The x-coordinate.

`y` - The y-coordinate.

Returns:

The brick at the specified position, or null if the specified position was empty.

See Also:

GameLogic

Class Brick

```
java.lang.Object
  └ Brick
```

```
public class Brick extends java.lang.Object
```

Class representing a Brick.

Field Summary

private int	**edges** An int representing where the Brick is connected to other Bricks.
private Player	**owner** The player that is or was in control of this Brick.
private int	**powerupType** The type of the powerup contained, 0 if none.

Constructor Summary

Brick(Player player, int edges) Constructor for Brick.	

Method Summary

int	**getEdges**() Gets the neighbors of this Brick.
Player	**getPlayer**() Retrieves the owner of this Brick.
int	**getPowerup**() Retrieves the powerup contained in this brick.
void	**removeEdges**(int edges) Removes edges from this Brick where a bit is set to 1.
void	**setPowerup**(int powerup)

	Sets the powerup contained in this Brick.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

owner

```
private Player owner
```

The player that is or was in control of this Brick.

---

edges

```
private int edges
```

An int representing where the Brick is connected to other Bricks. Each bit is 1 if the edge exists or 0 otherwise. The four least significant bits represent, from most significant to least significant: Top, right, down, left.

---

powerupType

```
private int powerupType
```

The type of the powerup contained, 0 if none.

Constructor Detail

Brick

```
public Brick(Playerplayer,
             intedges)
```

Constructor for Brick.

Parameters:

`player` - The player controlling the Brick.
`edges` - The neighbors of the brick.

Method Detail

setPowerup

```
public void setPowerup(intpowerup)
```

Sets the powerup contained in this Brick.

Parameters:
`powerup` - The powerup type, 0 if none.

---

getPowerup

```
public int getPowerup()
```

Retrieves the powerup contained in this brick.

Returns:
The type of the powerup contained, 0 if none.

---

getPlayer

```
public Player getPlayer()
```

Retrieves the owner of this Brick.

Returns:
Returns the player that owns the brick.

---

getEdges

```
public int getEdges()
```

Gets the neighbors of this Brick.

Returns:
Returns the number of edges with neighbours.

---

removeEdges

```
public void removeEdges(intedges)
```

Removes edges from this Brick where a bit is set to 1.

Parameters:
`edges` - The edges to remove.

Class CentralServer

```
java.lang.Object
```

└ **CentralServer**

---

public class **CentralServer**extends java.lang.Object

Class representing the central server.

Field Summary

private	**gameServers** Array of connection sockets, one per game server.
private	**serverList** List containing information about any game servers.
private ServerSocket	**serverSocket** A ServerSocket for listening to incoming connections.

Constructor Summary

CentralServer() Constructor for CentralServer.	

Method Summary

void	**addGameServer**(java.lang.String[] serverInfo) Adds a server to the game server list.
	getGameServers() Gets the list of game servers.
java.lang.String[]	**getServer**(java.lang.String name) Gets the game server with the given name.
void	**mainLoop**() The main loop, which is responsible for checking server availability (removes game servers that cannot be contacted.
void	**removeGameServer**(java.lang.String address) Removes a server from the game server list.
void	**resetTimeout**(java.lang.String address) Resets the timeout of the given server.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

serverSocket

private ServerSocket **serverSocket**

A ServerSocket for listening to incoming connections.

---

gameServers

`private` **`gameServers`**

Array of connection sockets, one per game server. These connections are closed as soon as the server has completed sending its information.

---

serverList

`private` **`serverList`**

List containing information about any game servers. Contains each server's name, address, the maximum number of players, information about whether the server is public or not (true or false), and the server's timeout. The format is as specified: [name address nrOfPlayers privateGame timeout]

Constructor Detail

CentralServer

`public` **`CentralServer`**`()`

Constructor for CentralServer.

Method Detail

mainLoop

`public void` **`mainLoop`**`()`

The main loop, which is responsible for checking server availability (removes game servers that cannot be contacted. The timeout is incremented for each game server in regular intervals, and any servers with too great a timeout are removed. It is each game server's responsiblity to contact this central server and reset the timeout.

---

resetTimeout

`public void` **`resetTimeout`**`(java.lang.Stringaddress)`

Resets the timeout of the given server.

Parameters:

`address` - The address of the game server whose timeout to reset.

---

getServer

```
public java.lang.String[] getServer(java.lang.Stringname)
```

Gets the game server with the given name.

Parameters:
`name` - The name of the game server to retrieve.

Returns:
The information of the game server with the given name, or null if it does not exist.

---

addGameServer

```
public void addGameServer(java.lang.String[]serverInfo)
```

Adds a server to the game server list.

Parameters:
`serverInfo` - A list of strings containing information about the server.

---

removeGameServer

```
public void removeGameServer(java.lang.Stringaddress)
```

Removes a server from the game server list.

Parameters:
`address` - The unique address of the server to be removed.

---

getGameServers

```
public  getGameServers()
```

Gets the list of game servers.

Returns:
A list of strings containing information about the server.

Class ClientCommunication

```
java.lang.Object
  └ClientCommunication
```

---

public class **ClientCommunication**extends java.lang.Object

A class for handling communication between the client and the GameServer or CentralServer.

Field Summary

static int	**CENTRAL_SERVER** An int representing the CentralServer.
private static java.lang.String	**CENTRAL_SERVER_ADDRESS** The address of the currently active CentralServer.
private Socket	**centralServerSocket** A socket connected to the current CentralServer.
static int	**GAME_SERVER** An int representing the GameServer.
private java.lang.String	**gameServereAddress** The address of the currently active GameServer.
private Socket	**gameServerSocket** A socket connected to the current GameServer.

Constructor Summary

ClientCommunication(java.lang.String centralServerAddress) Constructor for ClientCommunication.

Method Summary

java.lang.String[]	**getActions**(int server) Gets any new actions sent by a given server.
void	**sendAction**(java.lang.String action, int server) Sends an action to the current GameServer.
void	**setGameServer**(java.lang.String gameServerAddress) Sets the current GameServer.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

GAME_SERVER

public static final int **GAME_SERVER**

An int representing the GameServer.

CENTRAL_SERVER

```
public static final int CENTRAL_SERVER
```

An int representing the CentralServer.

CENTRAL_SERVER_ADDRESS

```
private static final java.lang.String CENTRAL_SERVER_ADDRESS
```

The address of the currently active CentralServer.

---

gameServereAddress

```
private java.lang.String gameServereAddress
```

The address of the currently active GameServer.

---

gameServerSocket

```
private Socket gameServerSocket
```

A socket connected to the current GameServer.

---

centralServerSocket

```
private Socket centralServerSocket
```

A socket connected to the current CentralServer.

Constructor Detail

ClientCommunication

```
public ClientCommunication(java.lang.StringcentralServerAddress)
```

Constructor for ClientCommunication.

Parameters:

`centralServerAddress` - The address of the current CentralServer.

Method Detail

setGameServer

```
public void setGameServer(java.lang.StringgameServerAddress)
```

Sets the current GameServer.

Parameters:
`gameServerAddress` - The address of the current GameServer.

---

sendAction

```
public void sendAction(java.lang.Stringaction,
                       intserver)
```

Sends an action to the current GameServer. This method is called by the doInputAction methods of the GameSessionState and LobbyState classes.

Parameters:
`action` - The action to send.
`server` - The server to send the action to. Should be either GAME_SERVER or CENTRAL_SERVER.
See Also:
GameSessionState, LobbyState

---

getActions

```
public java.lang.String[] getActions(intserver)
```

Gets any new actions sent by a given server. This method is called by the update loop of the ApplicationStateManager.

Parameters:
`server` - The server whose actions to get. Should be either GAME_SERVER or CENTRAL_SERVER.
See Also:
ApplicationStateManager

Class ControllerMap

```
java.lang.Object
  └ControllerMap
```

```
public class ControllerMapextends java.lang.Object
```

A class for mapping user input to in-game commands.

Constructor Summary

ControllerMap() Constructor for the ControllerMap.	

Method Summary

java.lang.String	**parseInput**(java.lang.String input) Parses an input string and converts it to an in-game action.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ControllerMap

public **ControllerMap**()

Constructor for the ControllerMap.

Method Detail

parseInput

public java.lang.String **parseInput**(java.lang.Stringinput)

Parses an input string and converts it to an in-game action. This method is called by getAction in the InputManager class.

Parameters:
input - The input to parse.

Returns:
An action depending on the input.

See Also:
InputManager

Class GameLogic

java.lang.Object
  └ **GameLogic**

```
public class GameLogicextends java.lang.Object
```

The GameLogic class, responsible for the game logic. For example, user commands are handled by the game logic, which then determines whether they are valid or not.

Field Summary

private Board	**board** The game Board used in the current game session.
private	**players** The list of players participating in the game.

Constructor Summary

GameLogic(players, int width, int height) The constructor for GameLogic.

Method Summary

void	**doActions**(java.lang.String[] actions) Performs a list of given actions in order from first to last.
private void	**dropPiece**(Player player) Moves the given player's piece downward as far as possible, and then fixates it.
Board	**getBoard**() Gets the game Board used in the current game session.
	getPlayers() Gets a list of the players participating in the game session.
private boolean	**movePiece**(Player player, int direction) Moves a player's piece in the specified direction.
private void	**rotatePiece**(Player player, int clockwise) Rotates the player's piece a given number of steps.
private void	**usePowerup**(Player player, int slot) Uses the powerup in the given slot, held by the given player.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

players

```
private  players
```

The list of players participating in the game.

---

board

```
private Board board
```

The game Board used in the current game session.

Constructor Detail

GameLogic

```
public GameLogic(players,
                 intwidth,
                 intheight)
```

The constructor for GameLogic.

Parameters:

`players` - The players participating in the game session.
`width` - The width of the Board.
`height` - The height of the Board.

Method Detail

getBoard

```
public Board getBoard()
```

Gets the game Board used in the current game session. This method is called by the render method of the GameSessionState class.

Returns:

the board

See Also:

`GameSessionState`

---

getPlayers

```
public  getPlayers()
```

Gets a list of the players participating in the game session.

Returns:

A list containing the participating players.

---

doActions

```
public void doActions(java.lang.String[]actions)
```

Performs a list of given actions in order from first to last. Each action is represented as a string. For example, rotating "player three"'s current piece counter-clockwise one step could be similar to "p3 ccw1". This method is called by doNetworkActions in GameSessionState. This method calls usePowerup, movePiece, rotatePiece, and

dropPiece.

Parameters:

`actions` - A list of strings representing the actions to perform.

See Also:

`GameSessionState`

---

usePowerup

```
private void usePowerup(Playerplayer,
                        intslot)
```

Uses the powerup in the given slot, held by the given player. This method is called by doActions.

Parameters:

`player` - The player that is holding the powerup to be used.

`slot` - The slot containing the powerup.

---

movePiece

```
private boolean movePiece(Playerplayer,
                          intdirection)
```

Moves a player's piece in the specified direction. The piece is only moved if the requested move is valid. This method is called by doActions.

Parameters:

`player` - The player whose piece to move.

`direction` - The direction in which to move the piece.

Returns:

True if the piece was successfully moved, false otherwise.

---

rotatePiece

```
private void rotatePiece(Playerplayer,
                         intclockwise)
```

Rotates the player's piece a given number of steps. This method is called by doActions, and in turn calls the rotate method of the Piece class.

Parameters:

`player` - The player whose piece to rotate.

`clockwise` - The number of steps to rotate in a clockwise direction. Negative values will rotate the piece counter-clockwise.

See Also:

`Piece`

---

dropPiece

```
private void dropPiece(Playerplayer)
```

Moves the given player's piece downward as far as possible, and then fixates it. This method is called by doActions.

Parameters:

`player` - The player whose piece to drop.

Class GameServer

```
java.lang.Object
  └GameServer
```

```
public class GameServerextends java.lang.Object
```

Class representing a game server. The class uses sockets to collect input from the clients, which are then sent with timestamps to each client. Each client is then responsible for determining whether a command is valid or not.

Field Summary

private	**actions** Array of actions to perform.
private boolean	**gameStarted** The gameStarted property.
private int	**numberOfPlayers** The number of players in the game.
private	**players** Array of connection sockets, one per player.
private ServerSocket	**serverSocket** A ServerSocket for listening to incoming connections.
private int	**timeStamp** The timeStamp property holds the current time stamp of the server.

Constructor Summary

GameServer(int numberOfPlayers) Constructor for GameServer.	

Method Summary

void	**mainLoop**() The main loop, responsible for collecting commands and activating sendPulse at given intervals.
void	**sendPulse**() Synchronizes the clients by sending collected commands, bundled with timestamps.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

serverSocket

```
private ServerSocket serverSocket
```

A ServerSocket for listening to incoming connections.

---

players

```
private  players
```

Array of connection sockets, one per player.

---

actions

```
private  actions
```

Array of actions to perform.

---

gameStarted

```
private boolean gameStarted
```

The gameStarted property. True when the game session is running, false otherwise.

---

numberOfPlayers

```
private int numberOfPlayers
```

The number of players in the game.

---

timeStamp

```
private int timeStamp
```

The timeStamp property holds the current time stamp of the server. It is incremented once each time a pulse is sent.

Constructor Detail

GameServer

```
public GameServer(intnumberOfPlayers)
```

Constructor for GameServer.

Parameters:

`numberOfPlayers` - The number of players to participate in the game.

Method Detail

mainLoop

```
public void mainLoop()
```

The main loop, responsible for collecting commands and activating sendPulse at given intervals.

---

sendPulse

```
public void sendPulse()
```

Synchronizes the clients by sending collected commands, bundled with timestamps. This method is called by mainLoop.

Class GameSessionState

```
java.lang.Object
  └GameSessionState
```

All Implemented Interfaces:

ApplicationState

```
public class GameSessionStateextends java.lang.Objectimplements ApplicationState
```

A class representing the game session state.

Field Summary

private java.lang.StringBuffer	**chatMessage** The chatMessage being written.
private boolean	**isChatting** The isChatting property, indicating whether the user is currently entering a chat message or not.
private GameLogic	**logic** The GameLogic instance used in this game session.
private ApplicationStateManager	**manager** The ApplicationStateManager controlling this state.

Constructor Summary

GameSessionState(ApplicationStateManager manager, players, int width, int height) The constructor for GameSessionState.

Method Summary

void	**doInputAction**(java.lang.String action) Updates the state according to user action.
void	**doNetworkActions**(java.lang.String[] actions) Updates the state according to the given server actions.
private void	**leaveState**() Leaves the GameSessionState and enters the MenuState.
void	**render**(Graphics g) Renders the current state.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

manager

private ApplicationStateManager **manager**

The ApplicationStateManager controlling this state.

---

logic

```
private GameLogic logic
```

The GameLogic instance used in this game session.

---

isChatting

```
private boolean isChatting
```

The isChatting property, indicating whether the user is currently entering a chat message or not.

---

chatMessage

```
private java.lang.StringBuffer chatMessage
```

The chatMessage being written.

Constructor Detail

GameSessionState

```
public GameSessionState(ApplicationStateManagermanager,
                        players,
                        intwidth,
                        intheight)
```

The constructor for GameSessionState.

Parameters:

`manager` - The manager of this state.
`players` - A list containing the different players to participate in the game session.
`width` - The width of the Board.
`height` - The height of the Board.

Method Detail

render

```
public void render(Graphicsg)
```

Renders the current state.

Specified by:

`render` in interface `ApplicationState`

Parameters:

`g` - The graphics context upon which to render the state.

---

doInputAction

```
public void doInputAction(java.lang.Stringaction)
```

Updates the state according to user action. This method calls sendAction in ClientCommunication unless isChatting is true, in which case chatMessage is updated.

Specified by:
doInputAction in interface ApplicationState

Parameters:
`action` - A String representing an action.

---

doNetworkActions

```
public void doNetworkActions(java.lang.String[]actions)
```

Updates the state according to the given server actions. This method calls the doActions method of the GameLogic class.

Parameters:
`actions` - An array of Strings representing actions to perform.

See Also:
GameLogic

---

leaveState

```
private void leaveState()
```

Leaves the GameSessionState and enters the MenuState.

Class InputManager

```
java.lang.Object
  └ InputManager
```

```
public class InputManagerextends java.lang.Object
```

A class that gathers user input, passes it through a ControllerMap, and then sends it to an ApplicationStateManager.

Field Summary

private int	**inputDeviceId** An integer representing the current input device.
private ApplicationStateManager	**manager** The ApplicationStateManager to recieve any input.
private ControllerMap	**map** Maps player input to in-game commands, for easy setup of multiple input configurations.

Constructor Summary

InputManager(ControllerMap cmap, int inputDeviceID) Constructor for the InputManager class.	

Method Summary

java.lang.String	**getAction**(java.lang.String input) Get the in-game action associated with the the given input.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

inputDeviceId

```
private int inputDeviceId
```

An integer representing the current input device.

---

manager

```
private ApplicationStateManager manager
```

The ApplicationStateManager to recieve any input.

---

map

```
private ControllerMap map
```

Maps player input to in-game commands, for easy setup of multiple input configurations.

Constructor Detail

InputManager

```
public InputManager(ControllerMapcmap,
                    intinputDeviceID)
```

Constructor for the InputManager class.

Parameters:

`cmap` - The ControllerMap to use in this InputManager.
`inputDeviceID` - The ID of the input device to use.

Method Detail

getAction

```
public java.lang.String getAction(java.lang.Stringinput)
```

Get the in-game action associated with the the given input. This method is called when input is gathered, and in turn calls the parseInput method of the ControllerMap class.

Parameters:

`input` - A string containing the user input.

Returns:

A string representing the associated action, as given by the ControllerMap.

See Also:

`ControllerMap`

Class LobbyState

```
java.lang.Object
  └ LobbyState
```

All Implemented Interfaces:

ApplicationState

```
public class LobbyStateextends java.lang.Objectimplements ApplicationState
```

Class representing a lobby state.Implements the ApplicationState interface. The lobby state contains the host and join multiplayer submenus together with a chat area.

Field Summary

private int	**activeWindow** The submenu currently beeing navigated.
private static int	**HOST_WINDOW**

	One possible submenu.
private boolean	**isHost** True if a game session is being hosted.
private static int	**JOIN_WINDOW** One possible submenu.
private static int	**LIST_WINDOW** One possible submenu.
private static int	**LOBBY_WINDOW** One possible submenu.
private ApplicationStateManager	**manager** The ApplicationStateManager controlling this state.
private int	**numberOfPlayers** The number of players set in a hosted game.

Constructor Summary

LobbyState(ApplicationStateManager manager) Constructor for LobbyState.	

Method Summary

private void	**cancel**() Goes one step back in the menu hierarchy.
private void	**createGame**(java.lang.String nickname, int numberOfPlayers, boolean privateGame) Creates a new game session.
void	**doInputAction**(java.lang.String action) Updates the state according to user action.
void	**doNetworkActions**(java.lang.String[] actions) Updates the state according to the given server actions.
void	**hostGame**() Sets up a hosted game.
private void	**joinGame**(java.lang.String address) Joins a hosted multiplayer game session.
private void	**launchGame**() Starts a multiplayer game session.
private void	**leaveState**(int newState) Leaves the LobbyState and enters the MenuState or GameSessionState.
void	**listGames**() Gets available games from CentralServer.
void	**render**(Graphics g) Renders the current state.
private void	**sendMessage**(java.lang.String message) Constructs a chat message to be sent to the server.
private void	**showMessage**(java.lang.String message) Displays a received chat message.
private void	**startMultiplayer**() Starts a multiplayer game session.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

HOST_WINDOW

```
private static final int HOST_WINDOW
```

One possible submenu.

See Also:
Constant Field Values

JOIN_WINDOW

```
private static final int JOIN_WINDOW
```

One possible submenu.

See Also:
Constant Field Values

LIST_WINDOW

```
private static final int LIST_WINDOW
```

One possible submenu.

See Also:
Constant Field Values

LOBBY_WINDOW

```
private static final int LOBBY_WINDOW
```

One possible submenu.

See Also:
Constant Field Values

---

manager

```
private ApplicationStateManager manager
```

The ApplicationStateManager controlling this state.

---

activeWindow

```
private int activeWindow
```

The submenu currently beeing navigated.

---

isHost

```
private boolean isHost
```

True if a game session is being hosted.

---

numberOfPlayers

```
private int numberOfPlayers
```

The number of players set in a hosted game.

Constructor Detail

LobbyState

```
public LobbyState(ApplicationStateManagermanager)
```

Constructor for LobbyState.

Parameters:

`manager` - The manager of this state.

Method Detail

render

```
public void render(Graphicsg)
```

Renders the current state.

Specified by:

`render` in interface `ApplicationState`

Parameters:

`g` - The graphics context upon which to render the state.

---

doInputAction

```
public void doInputAction(java.lang.Stringaction)
```

Updates the state according to user action.

Specified by:
`doInputAction` in interface `ApplicationState`

Parameters:
`action` - A String representing an action.

---

doNetworkActions

```
public void doNetworkActions(java.lang.String[]actions)
```

Updates the state according to the given server actions.

Parameters:
`actions` - An array of Strings representing actions to perform.

---

listGames

```
public void listGames()
```

Gets available games from CentralServer.

---

hostGame

```
public void hostGame()
```

Sets up a hosted game.

---

cancel

```
private void cancel()
```

Goes one step back in the menu hierarchy.

---

leaveState

```
private void leaveState(intnewState)
```

Leaves the LobbyState and enters the MenuState or GameSessionState.

Parameters:

`newState` - The state to enter.

---

showMessage

```
private void showMessage(java.lang.Stringmessage)
```

Displays a received chat message.

Parameters:

`message` - A String representing a chat message.

---

sendMessage

```
private void sendMessage(java.lang.Stringmessage)
```

Constructs a chat message to be sent to the server.

Parameters:

`message` - A String representing the message to be sent.

---

startMultiplayer

```
private void startMultiplayer()
```

Starts a multiplayer game session.

---

createGame

```
private void createGame(java.lang.Stringnickname,
                        intnumberOfPlayers,
                        booleanprivateGame)
```

Creates a new game session.

Parameters:

`nickname` - The nickname of the hosting player.
`numberOfPlayers` - The number of players allowed.
`privateGame` - True if this game session is to be hidden.

---

launchGame

```
private void launchGame()
```

Starts a multiplayer game session.

---

joinGame

```
private void joinGame(java.lang.Stringaddress)
```

Joins a hosted multiplayer game session.

Parameters:
`address` - The address to the host.

Class MenuState

```
java.lang.Object
  └MenuState
```

All Implemented Interfaces:
ApplicationState

```
public class MenuStateextends java.lang.Objectimplements ApplicationState
```

A class representing the menu state of the game.

Field Summary

private ApplicationStateManager	**manager** The ApplicationStateManager controlling this state.

Constructor Summary

MenuState(ApplicationStateManager manager) The constructor for MenuState.	

Method Summary

void	**doInputAction**(java.lang.String action) Performs a menu action depending on the action string provided.
private void	**exit**() Exits the game.
void	**render**(Graphics g) Renders the state.
private void	**showHelp**() Displays the help documention for the game.
private void	**showLobby**(boolean host) Switches to the lobby state.

private void	**startSingleplayer**() Starts a singleplayer game session.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

manager

```
private ApplicationStateManager manager
```

The ApplicationStateManager controlling this state.

Constructor Detail

MenuState

```
public MenuState(ApplicationStateManagermanager)
```

The constructor for MenuState.

Parameters:
`manager` - The ApplicationStateManager to control this state.

Method Detail

render

```
public void render(Graphicsg)
```

Renders the state.

Specified by:
`render` in interface `ApplicationState`
Parameters:
`g` - The graphics context upon which to render the state.

---

doInputAction

```
public void doInputAction(java.lang.Stringaction)
```

Performs a menu action depending on the action string provided.

Specified by:
`doInputAction` in interface `ApplicationState`
Parameters:
`action` - A string representing the action the user wants to perform.

showHelp

```
private void showHelp()
```

Displays the help documention for the game.

---

showLobby

```
private void showLobby(booleanhost)
```

Switches to the lobby state.

Parameters:

`host` - True if the user has requested to host a game, false if he wants to see a list of games.

---

exit

```
private void exit()
```

Exits the game.

---

startSingleplayer

```
private void startSingleplayer()
```

Starts a singleplayer game session.

Class Piece

```
java.lang.Object
  └ Piece
```

```
public class Pieceextends java.lang.Object
```

The Piece class, representing all the different piece types.

Field Summary

private Brick[][]	**matrix** An array containing the Bricks held in the current piece.
private char	**name** The name of the piece type.
private Player	**owner** The owner of this Piece.
private int	**rotationState** The current rotation state of the piece.

Constructor Summary

Piece(Player player, char pieceType) Constructor for Piece.	

Method Summary

boolean	**rotate**(int clockwise) Rotates the piece clockwise a given number of steps.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

matrix

```
private Brick[][] matrix
```

An array containing the Bricks held in the current piece.

---

name

```
private char name
```

The name of the piece type. Must be either I, J, L, O, S, T or Z.

---

rotationState

```
private int rotationState
```

The current rotation state of the piece.

owner

```
private Player owner
```

The owner of this Piece.

Constructor Detail

Piece

```
public Piece(Playerplayer,
             charpieceType)
```

Constructor for Piece.

Parameters:

`player` - The Player who controls this piece.
`pieceType` - The type of piece.

Method Detail

rotate

```
public boolean rotate(intclockwise)
```

Rotates the piece clockwise a given number of steps. This method is called by the rotatePiece method in the GameLogic class.

Parameters:

`clockwise` - The number of steps to rotate clockwise. Negative values rotates the piece counter-clockwise.

See Also:

GameLogic

Class PieceGenerator

```
java.lang.Object
  └ PieceGenerator
```

```
public class PieceGeneratorextends java.lang.Object
```

A class representing a piece generator. When instanced, the piece generator will create two bags that each hold one of each piece in random order. When a bag is empty the other bag will take over while the first one is refilled.

Field Summary

private Player	**myPlayer** The Player object controlling the PieceGenerator.
static java.util.ArrayList<java.util.ArrayList<Brick[][]>>	**pieceStates** A list representing the different possible pieces.
private Random	**rand** Used to generate random numbers internally.
private ShuffleBag[]	**s** A list of "bags" that hold randomly generated pieces.

Constructor Summary

PieceGenerator(Player player, int seed) Constructor for PieceGenerator.	

Method Summary

Piece	**getNext**() Get the next randomly generated piece.
Piece	**peekNext**(int offset) Looks at a future piece held in one of the ShuffleBags.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

pieceStates

```
public static java.util.ArrayList<java.util.ArrayList<Brick[][]>> pieceStates
```

A list representing the different possible pieces. The outer list holds different piece types, the inner list holds all possible rotations.

---

rand

```
private Random rand
```

Used to generate random numbers internally.

---

myPlayer

```
private Player myPlayer
```

The Player object controlling the PieceGenerator.

---

s

```
private ShuffleBag[] s
```

A list of "bags" that hold randomly generated pieces.

Constructor Detail

PieceGenerator

```
public PieceGenerator(Playerplayer,
                      intseed)
```

Constructor for PieceGenerator.

Parameters:

`player` - The player that this PieceGenerator belongs to.
`seed` - A random seed for this PieceGenerator.

Method Detail

getNext

```
public Piece getNext()
```

Get the next randomly generated piece.

Returns:

A random Piece.

---

peekNext

```
public Piece peekNext(intoffset)
```

Looks at a future piece held in one of the ShuffleBags.

Parameters:

`offset` - The offset index of the piece. The next piece is at index 0.

Returns:

The piece with the given offset in the list of upcoming pieces.

Class Player

```
java.lang.Object
  └ Player
```

```
public class Playerextends java.lang.Object
```

A class representing a player.

Field Summary

private Piece	**currentPiece** The piece that the player is currently controlling.
private int	**id** The player's id (to avoid name collisions).
private java.lang.String	**name** The name of the player.
private PieceGenerator	**pieceGenerator** A generator that generates random pieces for the player .
private int[]	**powerups** A list containing the player's powerups.

Constructor Summary

Player(java.lang.String name, int id, int slots, int seed) Constructor for Player.	

Method Summary

void	**addPowerup**(int powerup) Add a powerup to the player's list of powerups.
Piece	**getCurrentPiece**() Returns player's current piece.
int[]	**getPowerUps**() Get a list of the player's powerups.
void	**newPiece**() Generates a new piece and gives the player control of it.
int	**removePowerup**(int index) Remove a given powerup from the player's powerup list.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

currentPiece

```
private Piece currentPiece
```

The piece that the player is currently controlling.

---

powerups

```
private int[] powerups
```

A list containing the player's powerups. Each powerup is represented by an int, defining the type.

---

name

```
private java.lang.String name
```

The name of the player.

---

id

```
private int id
```

The player's id (to avoid name collisions).

---

pieceGenerator

```
private PieceGenerator pieceGenerator
```

A generator that generates random pieces for the player .

Constructor Detail

Player

```
public Player(java.lang.Stringname,
              intid,
              intslots,
              intseed)
```

Constructor for Player.

Parameters:

`name` - the name of the player.
`id` - the id of the player.

`slots` - the maximum number of powerups that can be held by this player.
`seed` - a random seed for the PieceGenerator.

Method Detail

getCurrentPiece

```
public Piece getCurrentPiece()
```

Returns player's current piece.

Returns:
The player's current piece.

---

getPowerUps

```
public int[] getPowerUps()
```

Get a list of the player's powerups.

Returns:
An int array containing the player's puwerups.

---

removePowerup

```
public int removePowerup(intindex)
```

Remove a given powerup from the player's powerup list.

Parameters:
`index` - The index of the powerup to be removed.
Returns:
An integer representing the removed powerup. If the supplied index is invalid -1 is returned.

---

addPowerup

```
public void addPowerup(intpowerup)
```

Add a powerup to the player's list of powerups.

Parameters:
`powerup` - An integer representing the type of powerup to add.

---

newPiece

```
public void newPiece()
```

Generates a new piece and gives the player control of it. This method is called from a

GameLogic doActions method when needed.

See Also:
GameLogic

Requirement references

Requirement Document Requirement	Design Document implementation
#1 Game pieces	The Piece and Brick classes.
#2 Boundaries	The Board and GameLogic classes (specifically, the movePiece method in the GameLogic class).
#3 Falling pieces	The Piece, GameLogic and GameServer classes (the GameLogic doNetworkActions recieves pulses from the GameServer when it is to move a Piece downward).
#4 Score	The GameLogic handles this in the various methods that move and rotate Pieces and Bricks.
#5 Get help	The MenuState class's showHelp method.
#6 Menu system	The MenuState class's render method shows the options, while input is handled by doActions.
#7 Start singleplayer game session	The MenuState class's startSingleplayer method.
#8 Host multiplayer game	The MenuState and LobbyState classes handle this. In MenuState the showLobby method is used, and in the LobbyState createGame and launchGame are used.
#9 Host private game	Same as above, with a String to indicate that the game is private sent to the CentralServer from the createGame method.
#10 Join multiplayer game	The MenuState and LobbyState classes handle this. In the MenuState the showLobby method is used, and in the LobbyState listGames is used.
#11 Chat with other users	The LobbyState and GameSessionState respectively handles this with doInputActions and doNetworkActions.
#12 Move piece vertically	The GameLogic class's dropPiece handles this.
#13 Move piece horizontally	The GameLogic class's movePiece handles this.
#14 Rotate piece	The GameLogic class's rotatePiece handles this.
#15 Collide with other player-controlled bricks	The GameLogic class's movePiece and dropPiece handle this.
#16 Piece control	The GameLogic class's doActions and the Player class's newPiece handle this.
#17 Fixate piece	The GameLogic class's movePiece and dropPiece and the Board class's fixPiece handle this.
#18 Remove row	The GameLogic class's movePiece and dropPiece and the Board class's removeRow handle this.
#19 Finishing a game session	The GameLogic class's createPiece handles this.
#20 Obtain powerup	The GameLogic class's doActions, the Board class's

	removeRow and the Player class's addPowerup handle this.
#21 Create powerup	The GameLogic class's doActions, the Board class's removeRow and getBrick, and the Brick class's setPowerup handle this (That is, when a row is removed, doActions will call getBrick and change that Brick to a powerup Brick using setPowerup).
#22 Use powerup	The GameLogic class's doActions and the Player class's removePowerup handle this.

Section 5.6