Before:

if (row >= rows.Length)

{

return false;

}

else if (rows[row] < num)

{

return false;

}

else

{

rows[row] -= num;

moveMade();

}

After:

if (row >= rows.Length || rows[row] < num || num <= 0)

{

return false;

}

else

{

rows[row] -= num;

moveMade();

}

Change: Simplified logic, added another check to make sure the move is valid.

Reason: Something

Before:

bool hasMoved = false;

while (!hasMoved){

int randRow = rand.Next(3);

if (game.rows[randRow] > 0){

int numRemoved = rand.Next(game.rows[randRow] - 1);

game.rows[randRow] -= numRemoved + 1;

hasMoved = true;

game.moveMade();

}

}

After:

bool hasMoved = false;

while (!hasMoved)

{

int randRow = rand.Next(3);

int numToRemove = rand.Next(game.rows[randRow]+1);

hasMoved = game.makeMove(randRow, numToRemove);

}

Change: Simplified logic, removed checking for 0. Removed call to game.moveMade and made moveMade private.

Reason: Something else.

Before:

public List<int> TopRow

{

get { return topRow; }

}

public List<int> MidRow

{

get { return midRow; }

}

public List<int> BotRow

{

get { return botRow; }

}

public int TotalStates{

get { return totalStates; }

}

public List<float> Values

{

get {

if (values.Count < topRow.Count){

calcValues();

}

return values;

}

}

After:

public IList<int> TopRow

{

get { return topRow; }

}

public IList<int> MidRow

{

get { return midRow; }

}

public IList<int> BotRow

{

get { return botRow; }

}

public int TotalStates{

get { return totalStates; }

}

public IList<float> Values

{

get {

if (values.Count < topRow.Count){

calcValues();

}

return values;

}

}

Change: Return ILists instead of lists.

Reason: Replacing Interfaces instead of collections, it allows calling code to be more flexible, even though it’s ridiculously negligible in this specific case.