/\*

\* Group Members: Hiroya Gojo, Zhongming Liao, Monty Choy Ziling Yu, Amy Petrine

\*/

// DO NOT CHANGE OR REMOVE THE FOLLOWING LINES

#ifndef \_\_DEFINE\_PLAYER\_FUNCTIONS\_CPP\_\_

#define \_\_DEFINE\_PLAYER\_FUNCTIONS\_CPP\_\_

#include <cstring>

#include <iomanip>

#include <iostream>

using namespace std;

#include "mapFunctions.cpp"

// DO NOT CHANGE OR REMOVE THE PRECEDING LINES

/\*

\* PLAYER STATE CONSTANTS

\*/

const char LOOKING\_UP = '^';

const char LOOKING\_DOWN = 'v';

const char LOOKING\_LEFT = '<';

const char LOOKING\_RIGHT = '>';

const int INVENTORY\_INDEX\_KEYS = 0;

const int INVENTORY\_INDEX\_PEBBLES = 1;

const int INVENTORY\_INDEX\_PLANK = 2;

const int INVENTORY\_INDEX\_ROPE = 3;

const int INVENTORY\_INDEX\_SLINGSHOT = 4;

const int INVENTORY\_LENGTH = 5;

int INVENTORY\_ARRAY[INVENTORY\_LENGTH] = { 0, 0, 0, 0, 0 };

/\*

\* PLAYER STATE

\*/

char playerSymbol;

int playerX, playerY;

/\*

\* FUNCTION PROTOTYPES

\*/

int getLookingAtX(const int, const char);

int getLookingAtY(const int, const char);

void inventoryAdd(const char);

bool inventoryHas(const char);

bool inventoryUse(const char);

void inventorySet(const char[], const int[], const int);

bool printInventoryRow(const int, const int);

int convertMapSquareToInventoryIndex(const char);

char convertInventoryIndexToItemChar(const int);

int max(int, int);

int min(int, int);

int getLookingAtX(const int currentX, const char currentSymbol)

{

switch(currentSymbol)

{

case LOOKING\_UP:

case LOOKING\_DOWN:

return currentX;

case LOOKING\_LEFT:

return (currentX - 1);

case LOOKING\_RIGHT:

return (currentX + 1);

}

return currentX;

}

int getLookingAtY(const int currentY, const char currentSymbol)

{

switch(currentSymbol) //Hiroya Gojo

{

case LOOKING\_UP:

return (currentY - 1);

case LOOKING\_DOWN:

return (currentY + 1);

case LOOKING\_LEFT:

case LOOKING\_RIGHT:

return currentY;

}

return currentY;

}

void inventoryAdd(const char item)

{

int inventoryIndex = convertMapSquareToInventoryIndex(item);

if (inventoryIndex >= 0)

{

switch(item)

{

//Hiroya Gojo

case MAP\_SQUARE\_KEY:

case MAP\_SQUARE\_PEBBLE:

INVENTORY\_ARRAY[inventoryIndex] -= 1;

case MAP\_SQUARE\_PEBBLES:

INVENTORY\_ARRAY[inventoryIndex] += 1;

case MAP\_SQUARE\_PLANK:

case MAP\_SQUARE\_ROPE:

INVENTORY\_ARRAY[inventoryIndex] += 1;

break;

case MAP\_SQUARE\_SLINGSHOT:

INVENTORY\_ARRAY[inventoryIndex] = 1;

break;

}

}

}

bool inventoryHas(const char item)

{

int inventoryIndex = convertMapSquareToInventoryIndex(item);

return (inventoryIndex >= 0 && INVENTORY\_ARRAY[inventoryIndex] > 0);

}

bool inventoryUse(const char item)

{

bool success = false;

int inventoryIndex = convertMapSquareToInventoryIndex(item);

if (inventoryIndex >= 0)

{

switch(inventoryIndex)

{

case INVENTORY\_INDEX\_KEYS:

case INVENTORY\_INDEX\_PEBBLES:

case INVENTORY\_INDEX\_PLANK:

case INVENTORY\_INDEX\_ROPE:

if (INVENTORY\_ARRAY[inventoryIndex] > 0)

{

INVENTORY\_ARRAY[inventoryIndex] -= 1;

success = true;

}

break;

case INVENTORY\_INDEX\_SLINGSHOT:

if (INVENTORY\_ARRAY[inventoryIndex] > 0)

{

success = true;

}

break;

}

}

return success;

}

void inventorySet(const char newInventoryItems[], const int newInventoryValues[], const int count)

{

for (int i = 0; i < INVENTORY\_LENGTH; i++)

{

bool itemIsInNewList = false;

for (int j = 0; j < count; j++)

{

int index = convertMapSquareToInventoryIndex(newInventoryItems[j]);

if (index == i)

{

INVENTORY\_ARRAY[index] = newInventoryValues[j];

itemIsInNewList = true;

break;

}

}

if (!itemIsInNewList)

{

INVENTORY\_ARRAY[i] = 0;

}

}

}

bool printInventoryRow(const int row, const int displayWidth)

{

bool success = false;

const int symbolWidth = max(0, min(2, displayWidth));

const int numberWidth = max(0, min(2, displayWidth - symbolWidth));

const int nameWidth = max(0, displayWidth - (numberWidth + symbolWidth));

switch(row)

{

case 0:

{

int itemIndex = INVENTORY\_INDEX\_KEYS;

int count = INVENTORY\_ARRAY[itemIndex];

if (count > 0)

{

//Monty Choy & Hiroya Gojo

cout << left << setw(symbolWidth) << convertInventoryIndexToItemChar(itemIndex)

<< left << setw(nameWidth) << "Key"

<< right << setw(numberWidth) << count;

success = true;

}

break;

}

case 1:

{

int itemIndex = INVENTORY\_INDEX\_ROPE;

int count = INVENTORY\_ARRAY[itemIndex];

if (count > 0)

{

//Monty Choy & Hiroya Gojo

cout << left << setw(symbolWidth) << convertInventoryIndexToItemChar(itemIndex)

<< left << setw(nameWidth) << "Rope"

<< right << setw(numberWidth) << count;

success = true;

}

break;

}

case 2:

{

int itemIndex = INVENTORY\_INDEX\_PLANK;

int count = INVENTORY\_ARRAY[itemIndex];

if (count > 0)

{

//Monty Choy & Hiroya Gojo

cout << left << setw(symbolWidth) << convertInventoryIndexToItemChar(itemIndex)

<< left << setw(nameWidth) << "Wood Plank"

<< right << setw(numberWidth) << count;

success = true;

}

break;

}

case 3:

{

break;

}

case 4:

{

bool hasSlingshot = INVENTORY\_ARRAY[INVENTORY\_INDEX\_SLINGSHOT] > 0;

int itemIndex = INVENTORY\_INDEX\_PEBBLES;

int count = INVENTORY\_ARRAY[itemIndex];

//Monty Choy & Hiroya Gojo

if (hasSlingshot)

{

itemIndex = INVENTORY\_INDEX\_SLINGSHOT;

cout << left << setw(symbolWidth) << convertInventoryIndexToItemChar(itemIndex)

<< left << setw(nameWidth) << "Slingshot"

<< right << setw(numberWidth) << count << left;

success = true;

}

else if (!hasSlingshot && count > 0)

{

cout << left << setw(symbolWidth) << convertInventoryIndexToItemChar(itemIndex)

<< left << setw(nameWidth) << "Pebble"

<< right << setw(numberWidth) << count << left;

success = true;

}

break;

}

}

return success;

}

int convertMapSquareToInventoryIndex(const char mapSquare)

{

switch(mapSquare)

{

case MAP\_SQUARE\_KEY:

return INVENTORY\_INDEX\_KEYS;

case MAP\_SQUARE\_PEBBLE:

case MAP\_SQUARE\_PEBBLES:

return INVENTORY\_INDEX\_PEBBLES;

case MAP\_SQUARE\_PLANK:

return INVENTORY\_INDEX\_PLANK;

case MAP\_SQUARE\_ROPE:

return INVENTORY\_INDEX\_ROPE;

case MAP\_SQUARE\_SLINGSHOT:

return INVENTORY\_INDEX\_SLINGSHOT;

}

return -1;

}

char convertInventoryIndexToItemChar(const int index)

{

switch(index)

{

case INVENTORY\_INDEX\_KEYS:

return MAP\_SQUARE\_KEY;

case INVENTORY\_INDEX\_PEBBLES:

return MAP\_SQUARE\_PEBBLE;

case INVENTORY\_INDEX\_PLANK:

return MAP\_SQUARE\_PLANK;

case INVENTORY\_INDEX\_ROPE:

return MAP\_SQUARE\_ROPE;

case INVENTORY\_INDEX\_SLINGSHOT:

return MAP\_SQUARE\_SLINGSHOT;

}

return ' ';

}

int max(int a, int b)

{

return (a > b ? a : b); //Monty Choy

}

int min(int a, int b)

{

return (a > b ? b : a); //Monty Choy

}

// DO NOT CHANGE OR REMOVE THE FOLLOWING LINE

#endif

// DO NOT CHANGE OR REMOVE THE PRECEDING LINE