/\*

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

\*/

// DO NOT CHANGE OR REMOVE THE FOLLOWING LINES

#ifndef \_\_DEFINE\_INTERACTION\_FUNCTIONS\_CPP\_\_

#define \_\_DEFINE\_INTERACTION\_FUNCTIONS\_CPP\_\_

#include <cctype>

#include <iostream>

using namespace std;

#include "playerFunctions.cpp"

#include "mapFunctions.cpp"

// DO NOT CHANGE OR REMOVE THE PRECEDING LINES

/\*

\* KEYBOARD CONSTANTS

\*/

const char KEYBOARD\_UP = 'w';

const char KEYBOARD\_DOWN = 's';

const char KEYBOARD\_LEFT = 'a';

const char KEYBOARD\_RIGHT = 'd';

const char KEYBOARD\_LOOK = 'l';

const char KEYBOARD\_TAKE = 't';

const char KEYBOARD\_USE = 'u';

const char KEYBOARD\_QUIT = 'Q';

const char KEYBOARD\_LOAD\_GAME = 'L';

const char KEYBOARD\_SAVE\_GAME = 'S';

/\*

\* FUNCTION PROTOTYPES

\*/

char readCharacterInput();

void doCommand(const char);

void doLook(const int, const int, const char);

void doTake(const int, const int, const char);

void doUse(const int, const int, const char);

void doLoadGame();

void doLoadGame(const char);

void doLoadDefaultGame();

void doUpdateAfterLoadGame();

void doSaveGame();

string lastMessage = "";

char readCharacterInput()

{

char input;

bool inputIsValid = false;

do

{

cin >> input;

cin.ignore(1024, '\n');

switch(input)

{

case KEYBOARD\_UP:

case KEYBOARD\_DOWN:

case KEYBOARD\_LEFT:

case KEYBOARD\_RIGHT:

case KEYBOARD\_LOOK:

case KEYBOARD\_TAKE:

case KEYBOARD\_USE:

case KEYBOARD\_QUIT:

case KEYBOARD\_LOAD\_GAME:

case KEYBOARD\_SAVE\_GAME:

inputIsValid = true;

break;

}

if (inputIsValid)

{

break;

}

cout << "Bad input. Try again." << endl;

} while (!inputIsValid);

return input;

}

void doCommand(const char command)

{

switch(command)

{

case KEYBOARD\_UP:

playerSymbol = LOOKING\_UP;

if (isOpenSpace(playerX, playerY - 1))

{

playerY -= 1;

}

break;

case KEYBOARD\_DOWN:

playerSymbol = LOOKING\_DOWN;

if (isOpenSpace(playerX, playerY + 1))

{

playerY += 1;

}

break;

case KEYBOARD\_LEFT: //Hiroya Gojo

playerSymbol = LOOKING\_LEFT;

if (isOpenSpace(playerX - 1, playerY))

{

playerX -= 1;

}

break;

case KEYBOARD\_RIGHT: //Hiroya Gojo

playerSymbol = LOOKING\_RIGHT;

if (isOpenSpace(playerX + 1, playerY))

{

playerX += 1;

}

break;

case KEYBOARD\_LOOK:

doLook(playerX, playerY, playerSymbol);

break;

case KEYBOARD\_TAKE:

doTake(playerX, playerY, playerSymbol); //Hiroya Gojo

break;

case KEYBOARD\_USE:

doUse(playerX, playerY, playerSymbol); //Hiroya Gojo

break;

case KEYBOARD\_LOAD\_GAME:

doLoadGame(); //Hiroya Gojo

break;

case KEYBOARD\_SAVE\_GAME:

doSaveGame(); //Hiroya Gojo

break;

}

}

void doLook(const int x, const int y, const char lookingDirection)

{

char mapSquare = getMapSquare(getLookingAtX(x, lookingDirection), getLookingAtY(y, lookingDirection));

switch(mapSquare)

{

case MAP\_SQUARE\_CHASM:

lastMessage = "The chasm in front of you is too wide to jump across. Perhaps there's another way across?";

break;

case MAP\_SQUARE\_EMPTY:

lastMessage = "You see nothing of interest.";

break;

case MAP\_SQUARE\_KEY:

lastMessage = "There is a shiny key on the ground. But what is it for?";

break;

case MAP\_SQUARE\_LOCK:

lastMessage = "The door in front of you is locked.";

break;

case MAP\_SQUARE\_PEBBLE:

lastMessage = "You see a large pebble on the ground. Stepping on it would hurt.";

break;

case MAP\_SQUARE\_PEBBLES:

lastMessage = "You see two large pebbles on the ground. Stepping on them would hurt.";

break;

case MAP\_SQUARE\_PLANK:

lastMessage = "There is a long plank of wood on the ground. You wonder how it got there.";

break;

case MAP\_SQUARE\_PLANK\_SET:

lastMessage = "The two sides of the chasm are bridged by a long plank of wood.";

break;

case MAP\_SQUARE\_ROPE:

lastMessage = "Someone left a long stretch of rope just lying around. How irresponsible.";

break;

case MAP\_SQUARE\_ROPE\_TIED:

lastMessage = "A rope dangles above the chasm in front of you. You can just barely reach it.";

break;

case MAP\_SQUARE\_ROCK:

lastMessage = "The rock wall in front of you is dusty with age. Try not to sneeze.";

break;

default:

lastMessage = "You're not sure what it is. You've never seen anything like it before.";

}

}

void doTake(const int x, const int y, const char lookingDirection)

{

bool success = false;

lastMessage = "There is nothing to take.";

int itemX = getLookingAtX(x, lookingDirection);

int itemY = getLookingAtY(y, lookingDirection);

char mapSquare = getMapSquare(itemX, itemY);

switch(mapSquare)

{

case MAP\_SQUARE\_KEY:

lastMessage = "You pick up the key.";

success = true;

break;

case MAP\_SQUARE\_PEBBLE:

lastMessage = "You pick up a pebble.";

success = true;

break;

case MAP\_SQUARE\_PEBBLES:

lastMessage = "You pick up a couple pebbles.";

success = true;

break;

case MAP\_SQUARE\_PLANK:

lastMessage = "You pick up a plank of wood.";

success = true;

break;

case MAP\_SQUARE\_ROPE:

lastMessage = "You pick up a long rope.";

success = true;

break;

case MAP\_SQUARE\_SLINGSHOT:

lastMessage = "You pick up a slingshot.";

success = true;

break;

}

if (success)

{

inventoryAdd(mapSquare);

clearMapSquare(itemX, itemY, mapSquare);

}

}

void doUse(const int x, const int y, const char lookingDirection)

{

char itemToUse;

cout << endl << "What would you like to use? ";

cin >> itemToUse;

cin.ignore(1024, '\n');

if (!(inventoryHas(itemToUse))) //Hiroya Gojo

{

lastMessage = "You don't have any.";

return;

}

int itemX = getLookingAtX(x, lookingDirection);

int itemY = getLookingAtY(y, lookingDirection);

char mapSquare = getMapSquare(itemX, itemY); //Hiroya Gojo

if (mapSquare == MAP\_SQUARE\_CHASM && itemToUse == MAP\_SQUARE\_PLANK)

{

inventoryUse(itemToUse);

setMapSquare(itemX, itemY, mapSquare, MAP\_SQUARE\_PLANK\_SET);

lastMessage = "You lay the plank of wood over the chasm. It just barely touches both sides.";

}

else if (mapSquare == MAP\_SQUARE\_CHASM && itemToUse == MAP\_SQUARE\_ROPE)

{

inventoryUse(itemToUse);

setMapSquare(itemX, itemY, mapSquare, MAP\_SQUARE\_ROPE\_TIED);

lastMessage = "Standing on the tips of your toes, you reach up and tie the rope to a beam above you.";

}

else if (mapSquare == MAP\_SQUARE\_LOCK && itemToUse == MAP\_SQUARE\_KEY)

{

inventoryUse(itemToUse);

clearMapSquare(itemX, itemY, mapSquare);

lastMessage = "You turn the key. Hard. Just as the lock opens you feel the key snap in half.";

}

else if (mapSquare == MAP\_SQUARE\_CHASM && itemToUse == MAP\_SQUARE\_PEBBLE) //Hiroya Gojo Num25

{

inventoryUse(itemToUse);

lastMessage = "You drop a pebble into the chasm, counting the seconds until it hits the bottom. You hear nothing.";

}

else if (mapSquare == MAP\_SQUARE\_PLANK && itemToUse == MAP\_SQUARE\_PEBBLE) //Hiroya Gojo Num26

{

inventoryUse(itemToUse);

inventoryHas(MAP\_SQUARE\_SLINGSHOT);

setMapSquare(itemX, itemY, mapSquare, MAP\_SQUARE\_PEBBLE);

lastMessage = "You shoot the plank of wood with your slingshot. No more plank.";

}

else

{

lastMessage = "You can't use that here.";

}

}

void doLoadGame()

{

cout << endl << "Enter 0-9 to load a saved game. Enter any other visible character to cancel. Input: ";

char slot;

cin >> slot;

doLoadGame(slot);

}

void doLoadGame(const char slot)

{

if (slot > '9' || slot < '0' || !isdigit(slot)) //Hiroya Gojo

{

lastMessage = "Must enter 0-9 to load a saved game.";

return;

}

//Monty Choy

string fileName = "gameSlot0.txt"; //default file name

//const char \* slot2 = new char(slot); //convert slot so it can be passed in to String class methods

string slot2(1, slot); //Hiroya Gojo

fileName.replace(fileName.find("0"), 1, slot2); //replace '0' int fileName with slot

if (!loadGame(fileName))

{

lastMessage.append("Could not load '");

lastMessage.append(fileName);

lastMessage.append("'. File is corrupt or does not exist.");

return;

}

doUpdateAfterLoadGame();

lastMessage.append("Loaded game ");

lastMessage.append(slot2);

lastMessage.append(".");

}

void doLoadDefaultGame()

{

loadDefaultMap(); //Hiroya Gojo

doUpdateAfterLoadGame();

lastMessage = "Loaded default game.";

}

void doUpdateAfterLoadGame()

{

playerX = FILE\_START\_LOCATION\_AS\_INDEX % MAP\_WIDTH;

playerY = FILE\_START\_LOCATION\_AS\_INDEX / MAP\_WIDTH;

playerSymbol = ((FILE\_START\_LOOKING\_DIRECTION == LOOKING\_UP ||

FILE\_START\_LOOKING\_DIRECTION == LOOKING\_DOWN ||

FILE\_START\_LOOKING\_DIRECTION == LOOKING\_LEFT ||

FILE\_START\_LOOKING\_DIRECTION == LOOKING\_RIGHT) ? FILE\_START\_LOOKING\_DIRECTION : LOOKING\_UP);

inventorySet(FILE\_INVENTORY\_ITEMS, FILE\_INVENTORY\_VALUES, FILE\_INVENTORY\_LENGTH);

}

void doSaveGame()

{

cout << endl << "Enter 0-9 to save the game in that slot. Enter any other visible character to cancel. Input: ";

char slot;

cin >> slot;

if (slot > '9' || slot < '0' || !isdigit(slot)) //Hiroya Gojo

{

lastMessage = "Must enter 0-9 to load a saved game.";

return;

}

//Monty Choy

string fileName = "gameSlot0.txt"; //default file name

//const char \* slot2 = new char(slot); //convert slot so it can be passed in to String class methods

string slot2(1, slot); //Hiroya Gojo

fileName.replace(fileName.find("0"), 1, slot2); //replace '0' int fileName with slot

int numberOfItems = 0;

char inventoryItems[INVENTORY\_LENGTH];

int inventoryValues[INVENTORY\_LENGTH];

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

{

int itemCount = INVENTORY\_ARRAY[i];

if (itemCount > 0)

{

inventoryItems[numberOfItems] = convertInventoryIndexToItemChar(i);

inventoryValues[numberOfItems] = itemCount;

numberOfItems++;

}

}

if (!saveGame(fileName, playerX, playerY, playerSymbol, inventoryItems, inventoryValues, numberOfItems))

{

lastMessage.append("Could not save '");

lastMessage.append(fileName);

lastMessage.append("'. Data is corrupt or writing files is not allowed.");

return;

}

lastMessage.append("Saved game to slot ");

lastMessage.append(slot2);

lastMessage.append(".");

}

// DO NOT CHANGE OR REMOVE THE FOLLOWING LINE

#endif

// DO NOT CHANGE OR REMOVE THE PRECEDING LINE