Zachary Briggs

October 11th 2017

Assessment One

**1: Requirements**

1.1

                    Name: Text Based Adventure Game

Problem Statement: Create a text based adventure game in C++

Problem Specifics: Use of the string class created for this assessment as well as class inheritance that is at least 2 levels deep.

**1.2 Input Information**

Enter ‘w’ to go north/add (-1,0) to mPosition, ‘s’ to go south/add (1,0) to mPosition, ‘a’ to go west/add (0, -1) to mPosition, ‘d’ to go south/add (0,1) to mPosition.

**1.3 Output Information**

**1.4 User Interface**

**2: System Architecture**

**Dungeon.h**

Prototype: bool CheckPlayerPosition();

Description: Checks a Player’s position and returns true if the player

is outside the dungeon limitations otherwise it returns false.

Arguments: None.

Precondition: None.

Postcondition: True/False is returned.

Protection: Public.

Prototype: void PrintRooms();

Description: Prints out each individual room in a dungeon.

Arguments: None.

Precondition: None.

Postcondition: Each room is printed.

Protection: Public.

Prototype: void GenRooms();

Description: Generates the rooms of a dungeon.

Arguments: None.

Precondition: None.

Postcondition: A new dungeon is generated.

Protection: Public.

**Entity.h**

/\*Prototype: void SetPosition(Point2D\* pos);

Description: Sets the player's position to a chosen position.

Arguments: A Point2D representing the new position.

Precondition: None.

Postcondition: The players mPosition is changed to the position passed in.

Protection: Public.

Prototype: Point2D GetPosition();

Description: Returns the player's mPosition.

Arguments: None.

Precondition: None.

Postcondition: mPosition is returned.

Protection: Public.

**Game.h**

Prototype: void PlayerMove();

Description: Asks the player for input and then calls the Player's Move function

Arguments: None.

Precondition: The player is at a position.

Postcondition: The player's position is changed depending on their input

Protection: Public

Prototype: bool HazardCheck();

Description: Checks if the player is on top of or nearby the Wumpus,gold, or a trap.

Arguments: None.

Precondition: The player is on or near Wumpus,gold, or a trap.

Postcondition: Text is outputted if the player is nearby an object or the game ends if

they're on top of it.

Protection: Public.

Prototype: bool PlayAgain();

Description: Asks the player if they want to play again after the game has finished.

Returns true or false depending on their answer.

Arguments: None.

Precondition: The player completes the game and the player is asked if they wish to

play again.

Postcondition: The game is restarted or the application exits.

Protection: Public.

**Item.h**

Prototype: Point2D GetPosition();

Description: Returns the Item's mPosition.

Arguments: None.

Precondition: None

Postcondition: Item's mPosition is returned.

Protection: Public

Prototype: bool CheckForPlayer(Entity\* dude);

Description: Checks to see an Entity is occupying the same position as the Item.

Arguments: An entity.

Precondition: None.

Postcondition: Returns true if an Entity is occupying the same space. Returns false

if it isn't.

Protection: Public

Prototype: bool IsPlayerNearby(Entity\* dude);

Description: Checks to see if an Entity is in an adjacent position.

Arguments: The player.

Precondition: None

Postcondition: Returns true if an Entity is occupying an adjacent position. Returns

false if they aren't.

Protection: Public

**Player.h**

Prototype: void Move(char direction);

Description: Takes user's input and changes the Player's mPosition depending on the

input.

Arguments: A char representing which direction the player wants to go.

Precondition: None

Postcondition: The Player's mPosition is changed depending on the input.

Protection: Public

Prototype: void SetPosition(Point2D\* pos);

Description: Changes the player's mPosition to the value passed in.

Arguments: A Point2D representing the new mPosition.

Precondition: None

Postcondition: Player's mPosition is changed to the value passed in.

Protection: Public

Prototype: Point2D GetPosition();

Description: Returns the Player's mPosition.

Arguments: None.

Precondition: None

Postcondition: Player's mPosition is returned.

Protection: Public

**Point2D.h**

Prototype: Point2D operator+(const Point2D & other);

Description: Adds two Point2D variables and returns the value.

Arguments: Two Point2D variables.

Precondition: Two Point2D variables.

Postcondition: The 2 Point2Ds are added and returned as a new

temp Point2D. Neither variable is changed.

Visibility: Public.

Prototype: Point2D operator-(const Point2D & other);

Description: Subtracts two Point2D variables and returns the value.

Arguments: Two Point2D variables.

Precondition: Two Point2D variables.

Postcondition: The 2 Point2Ds are subtracted and returned as a new

temp Point2D. Neither variable is changed.

Visibility: Public.

Prototype: Point2D operator\*(const Point2D & other);

Description: Multiplies two Point2D variables and returns the value.

Arguments: Two Point2D variables.

Precondition: Two Point2D variables.

Postcondition: The 2 Point2Ds are multiplied and returned as a new

temp Point2D. Neither variable is changed.

Visibility: Public.

Prototype: Point2D operator-(float other);

Description: Both variables in a Point2D are multiplied by one variable.

Precondition: One Point2D variables.

Postcondition: Both Point2D variables are multiplied by other and

are returned as a temp value. Neither variable is changed.

Visibility: Public.

Prototype: Point2D operator+=(const Point2D & other);

Description: Adds two Point2D variables.

Arguments: Two Point2D variables.

Precondition: Two Point2D variables.

Postcondition: The 2 Point2Ds are added and the values of the first

Point2D are changed.

Visibility: Public.

Prototype: Point2D operator-=(const Point2D & other);

Description: Subtracts two Point2D variables.

Arguments: Two Point2D variables.

Precondition: Two Point2D variables.

Postcondition: The 2 Point2Ds are subtracted and the values of the

first Point2D are changed.

Visibility: Public.

Prototype: bool operator==(const Point2D & other);

Description: Returns true or false depending on if the two Point2Ds

are equal.

Arguments: Two Point2Ds

Precondition: Two Point2Ds

Postcondition: True or false is returned depending on if the Point2Ds

are equal.

Visibility: Public.

Prototype: friend std::ostream& operator<<(std::ostream&stream, const Point2D & a);

Description: Outputs mX and mY of a Point2D.

Arguments: A Point2D

Precondition: A Point2D

Postcondition: mX and mY of a Point2D are outputted to the console.

Visibility: Public.

Prototype: float GetX();

Description: Returns mX of a Point2D.

Arguments: A Point2D.

Precondition: A Point2D.

Postcondition: mX of the Point2D is returned.

Visibility: Public.

Prototype: float GetY();

Description: Returns mY of a Point2D.

Arguments: A Point2D.

Precondition: A Point2D.

Postcondition: mY of the Point2D is returned.

Visibility: Public.

**Wumpus.h**

Prototype: void SetPosition(Point2D\* pos);

Description: Changes the Wumpus' mPosition to the value passed in.

Arguments: A Point2D representing the new mPosition.

Precondition: None

Postcondition: Wumpus' mPosition is changed to the value passed in.

Protection: Public

Prototype: Point2D GetPosition();

Description: Returns the Wumpus' mPosition.

Arguments: None.

Precondition: None

Postcondition: The Wumpus' mPosition is returned.

Protection: Public

Prototype: bool CheckForPlayer(Player\* dude);

Description: Checks to see if the player is occupying the same position as the Wumpus.

Arguments: The player.

Precondition: None.

Postcondition: Returns true if the player is occupying the same space. Returns false

if they aren't.

Protection: Public

Prototype: bool IsPlayerNearby(Player\* dude);

Description: Checks to see if the player is in an adjacent position.

Arguments: The player.

Precondition: None

Postcondition: Returns true if the player is occupying an adjacent position. Returns

false if they aren't.

Protection: Public

**3: Source Code**

**Dungeon.cpp**

#include "Dungeon.h"

Dungeon::Dungeon()

{

mRooms = new Point2D[25];

mNumCols = 5;

mNumRows = 5;

}

Dungeon::Dungeon(int rows, int cols)

{

mRooms = new Point2D[rows \* cols];

mNumCols = cols;

mNumRows = rows;

}

void Dungeon::GenRooms()

{

int iter = 0;

for (int i = 0; i < mNumRows; i++)

{

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

{

mRooms[iter] = Point2D(i, j);

iter++;

}

}

}

bool Dungeon::CheckPlayerPosition(Player \*player)

{

for (int i = 0; i <= mNumRows+1; i++)

{

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

{

Point2D thing(i, j);

if (player->GetPosition() == thing)

return true;

}

}

return false;

}

void Dungeon::PrintRooms()

{

GenRooms();

for (int i = 0; i <= mNumRows; i++)

{

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

{

std::cout << "<" << i << "," << j << ">";

if (j == mNumCols)

std::cout << std::endl;

}

}

}

**Entity.cpp**

#include "Entity.h"

Entity::Entity()

{

mPosition = new Point2D();

}

void Entity::SetPosition(Point2D\* pos)

{

mPosition = pos;

}

Point2D Entity::GetPosition()

{

return \*mPosition;

}

**Game.cpp**

#include "Game.h"

Game::Game()

{

mPlayer = new Player();

mWumpus = new Wumpus();

mDungeon = new Dungeon(5, 5);

mTraps = new Item[3];

mTraps[0] = Item(new Point2D(1,4));

mTraps[1] = Item(new Point2D(3,3));

mTraps[2] = Item(new Point2D(5,1));

mGold = new Item(new Point2D(2, 4));

mDungeon->PrintRooms();

}

void Game::PlayerMove()

{

char direction;

std::cout << "Choose a direction.\n";

std::cin >> direction;

mPlayer->Move(direction);

}

bool Game::HazardCheck()

{

if (mDungeon->CheckPlayerPosition(mPlayer) == false)

{

std::cout << "You stepped in the poison swamp!\n";

return false;

}

if (mWumpus->CheckForPlayer(mPlayer) == true)

{

std::cout << "The Wumpus found you!\n";

return false;

}

if (mWumpus->IsPlayerNearby(mPlayer) == true)

{

std::cout << "You smell a foul stench...\n";

}

if (mTraps[0].CheckForPlayer(mPlayer) || mTraps[1].CheckForPlayer(mPlayer) || mTraps[2].CheckForPlayer(mPlayer))

{

std::cout << "You fell into a pit!\n";

return false;

}

if (mTraps[0].IsPlayerNearby(mPlayer) || mTraps[1].IsPlayerNearby(mPlayer) || mTraps[2].IsPlayerNearby(mPlayer))

{

std::cout << "You feel a breeze nearby...\n";

}

if (mGold->CheckForPlayer(mPlayer) == true)

{

std::cout << "You found the gold!\n";

return false;

}

if (mGold->IsPlayerNearby(mPlayer) == true)

std::cout << "You see a faint shimmering...\n";

return true;

}

bool Game::PlayAgain()

{

char answer;

std::cout << "Would you like to play again? y/n\n";

std::cin >> answer;

if (answer = 'y')

return true;

else if (answer = 'n')

return false;

}

**Item.cpp**

#include "Item.h"

Item::Item()

{

mPosition = new Point2D(4, 4);

}

Item::Item(Point2D \*pos)

{

mPosition = pos;

}

Point2D Item::GetPosition()

{

return \*mPosition;

}

bool Item::CheckForPlayer(Entity \* dude)

{

if (dude->GetPosition() == \*mPosition)

{

return true;

}

return false;

}

bool Item::IsPlayerNearby(Entity \* dude)

{

for (int i = -1; i < 2; i += 2)

{

if ((GetPosition() + Point2D(0, i)) == dude->GetPosition())

return true;

}

for (int j = -1; j < 2; j += 2)

{

if ((GetPosition() + Point2D(j, 0)) == dude->GetPosition())

return true;

}

return false;

}

**Player.cpp**

#include "Player.h"

Player::Player()

{

mPosition = new Point2D(4,2);

}

void Player::Move(char direction)

{

if (direction == 'w')

\*mPosition += \*new Point2D(-1, 0);

if (direction == 's')

\*mPosition += \*new Point2D(1, 0);

if (direction == 'a')

\*mPosition += \*new Point2D(0, -1);

if (direction == 'd')

\*mPosition += \*new Point2D(0, 1);

}

void Player::SetPosition(Point2D \* pos)

{

Entity::SetPosition(pos);

}

Point2D Player::GetPosition()

{

return \*mPosition;

}

Point2D.cpp

#include "Point2D.h"

Point2D::Point2D()

{

mX = 1;

mY = 1;

}

Point2D::Point2D(int x, int y)

{

mX = x;

mY = y;

}

Point2D Point2D::operator+(const Point2D & other)

{

Point2D temp;

temp.mX = mX + other.mX;

temp.mY = mY + other.mY;

return Point2D(temp);

}

Point2D Point2D::operator-(const Point2D & other)

{

Point2D temp;

temp.mX = mX - other.mX;

temp.mY = mY - other.mY;

return Point2D(temp);

}

Point2D Point2D::operator\*(const Point2D & other)

{

Point2D temp;

temp.mX = mX \* other.mX;

temp.mY = mY \* other.mY;

return Point2D(temp);

}

Point2D Point2D::operator\*(int other)

{

Point2D temp;

temp.mX = mX \* other;

temp.mY = mY \* other;

return Point2D(temp);

}

Point2D Point2D::operator+=(const Point2D & other)

{

mX = mX += other.mX;

mY = mY += other.mY;

return Point2D();

}

Point2D Point2D::operator-=(const Point2D & other)

{

mX = mX -= other.mX;

mY = mY -= other.mY;

return Point2D();

}

bool Point2D::operator==(const Point2D & other)

{

return (mX == other.mX && mY == other.mY);

}

int Point2D::GetX()

{

return mX;

}

int Point2D::GetY()

{

return mY;

}

std::ostream & operator<<(std::ostream & stream, const Point2D & a)

{

stream << a.mX;

std::cout << ",";

stream << a.mY;

return stream;

}

**Wumpus.cpp**

#include "Wumpus.h"

#include "Point2D.h"

Wumpus::Wumpus()

{

mPosition = new Point2D(2, 1);

}

void Wumpus::SetPosition(Point2D \* pos)

{

Entity::SetPosition(pos);

}

Point2D Wumpus::GetPosition()

{

return \*mPosition;

}

bool Wumpus::CheckForPlayer(Player\* dude)

{

if (dude->GetPosition() == \*mPosition)

{

return true;

}

return false;

}

bool Wumpus::IsPlayerNearby(Player\* dude)

{

for (int i = -1; i < 2; i+=2)

{

if (((GetPosition() + Point2D(0, i)) == dude->GetPosition()))

{

return true;

}

}

for (int j = -1; j < 2; j += 2)

{

if (((GetPosition() + Point2D(j, 0)) == dude->GetPosition()))

{

return true;

}

}

return false;

}

**Main.cpp**

#include "Game.h"

int main()

{

Game\* newGame = new Game();

while (newGame->HazardCheck())

{

newGame->PlayerMove();

}

std::cout << "Game Over.\n";

system("pause");

}

**4: Read Me**