Zachary Briggs

October 11th 2017

Assessment One

**1: Requirements**

**1.1 Goal**

**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**

Outputs text depending on if the player is near or in the same position as Wumpus, a pitfall, or the gold. Will also output “Game Over” once the game is finished regardless of if the player won or not.

**1.4 User Interface**

N/A

**2: System Architecture**

**Dungeon.h**

Prototype: Dungeon Dungeon();

Description: Initilizes a dungeon with default mRooms,mNumCols, and mNumRows.

Arguments: None.

Precondition: None.

Postcondition: A dungeon with default values is created.

Protection: Public.

Prototype: Dungeon ~Dungeon();

Description: Deletes a Dungeon and all the variables within.

Arguments: None.

Precondition: A dungeon is created.

Postcondition: The dungeon is deleted.

Protection: Public.

Prototype: Dungeon Dungeon(int rows, int cols);

Description: Initilizes a dungeon with default mRooms, but mNumCols and mNumrows

are initilized with the values passed in.

Arguments: None.

Precondition: None.

Postcondition: A dungeon with custom mNumRows and mNumCols is created.

Protection: Public.

Prototype: bool CheckPlayerPosition();

Description: Checks each individual position inside of a dungeon and if they player

isn't at any of the positions, false is returned.If the player is found then true is

returned.

Arguments: None.

Precondition: None.

Postcondition: True/False is returned.

Protection: Public.

Prototype: void GenRooms();

Description: Assigns values to every room in a dungeon.

Arguments: None.

Precondition: None.

Postcondition: A new dungeon is generated.

Protection: Public.

Prototype: void DungeonBorderWarning(Player\* player);

Description: Warns the player when they are on the border of the map.

Arguments: A player.

Precondition: A player is on the border of the map.

Postcondition: Text is outputted telling the player they're on the border of the map.

Protection: Public.

**Entity.h**

Prototype: Entity Entity();

Description: Creates an Entity with a default mPosition.

Arguments: None.

Precondition: None.

Postcondition: An default Entity is created.

Protection: Public.

Prototype: Entity ~Entity();

Description: Deletes an Entity.

Arguments: None.

Precondition: An Entity is created.

Postcondition: The Entity is deleted.

Protection: Public.

Prototype: void SetPosition(Point2D\* pos);

Description: Sets the player's mPosition to a a Point2D passed in.

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: Game Game();

Description: Creates a Game with default mPlayer, mWumpus, mDungeon, mTraps, and

mGold.

Arguments: None.

Precondition: None.

Postcondition: A default Game is created.

Protection: Public.

Prototype: Game ~Game();

Description: Deletes a Game and all the variables within it.

Arguments: None.

Precondition: A Game is created.

Postcondition: The Game is deleted.

Protection: Public.

Prototype: void TitleScreen();

Description: Outputs title and controls to the console.

Arguments: None.

Precondition: None.

Postcondition: The title and controls are displayed in the console.

Protection: Public.

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: True/False is returned.

Protection: Public.

**Item.h**

Prototype: Item Item();

Description: An item with a default mPosition is created.

Arguments: None.

Precondition: None.

Postcondition: A default Item is created.

Protection: Public.

Prototype: Item ~Item();

Description: Deletes an Item.

Arguments: None.

Precondition: An Item is created.

Postcondition: The Item is deleted

Protection: Public.

Prototype: Item(Point2D \*pos);

Description: An Item with a custom mPosition is created.

Arguments: None.

Precondition: None.

Postcondition: A custom Item is created.

Protection: Public.

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\* entity);

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\* entity);

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: Player Player();

Description: Creates a Player with a default mPosition.

Arguments: None.

Precondition: None.

Postcondition: A player with a default mPosition is created.

Protection: Public.

Prototype: Player ~Player();

Description: Deletes the Player.

Arguments: None.

Precondition: A Player is created.

Postcondition: The Player is deleted.

Protection: Public.

Prototype: void Move(String direction);

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

input.

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

Precondition: None

Postcondition: The Player's mPosition is changed dependning 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 Point2D();

Description: A Point2D with default mX and mY is created.

Arguments: None.

Precondition: None.

Postcondition: A default Point2D is created.

Protection: Public.

Prototype: Point2D ~Point2D();

Description: A Point2D is deleted.

Arguments: None.

Precondition: A Point2D is created.

Postcondition: The Point2D is deleted.

Protection: Public.

Prototype: Point2D Point2D(int x, int y);

Description: A Point2D with custom mX and mY is created.

Arguments: None.

Precondition: None.

Postcondition: A custom Point2D is created.

Protection: Public.

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: 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: 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: Wumpus Wumpus();

Description: A Wumpus with default mPosition is opened.

Arguments: None.

Precondition: None.

Postcondition: A default Wumpus is created.

Protection: Public.

Prototype: Wumpus ~Wumpus();

Description: A Wumpus is deleted.

Arguments: None.

Precondition: A Wumpus is created.

Postcondition: The Wumpus is now deleted.

Protection: Public.

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\* player);

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

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\* player);

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.h**

#pragma once

#include "Player.h"

class Dungeon

{

private:

Point2D\* mRooms;

int mNumCols;

int mNumRows;

public:

Dungeon();

~Dungeon();

Dungeon(int rows, int cols);

bool CheckPlayerPosition(Player \*player);

void GenRooms();

bool DungeonBorderWarning(Player\* player);

};

**Dungeon.cpp**

#include "Dungeon.h"

Dungeon::Dungeon()

{

mRooms = new Point2D[25];

mNumCols = 5;

mNumRows = 5;

}

Dungeon::~Dungeon()

{

}

Dungeon::Dungeon(int rows, int cols)

{

mRooms = new Point2D[rows \* cols];

mNumCols = cols;

mNumRows = rows;

}

void Dungeon::GenRooms()

{

int iter = 0;

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

{

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

{

mRooms[iter] = Point2D(x, y);

iter++;

}

}

}

bool Dungeon::DungeonBorderWarning(Player\* player)

{

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

{

if (player->GetPosition() == Point2D(0, w))

{

std::cout << "You hear a faint bubbling...\n";

return true;

}

}

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

{

if (player->GetPosition() == Point2D(x, 4))

{

std::cout << "You hear a faint bubbling...\n";

return true;

}

}

for (int y = 4; y >= 0; y -= 1)

{

if (player->GetPosition() == Point2D(4, y))

{

std::cout << "You hear a faint bubbling...\n";

return true;

}

}

for (int z = 4; z >= 0; z -= 1)

{

if (player->GetPosition() == Point2D(z, 0))

{

std::cout << "You hear a faint bubbling...\n";

return true;

}

}

return false;

}

bool Dungeon::CheckPlayerPosition(Player \*player)

{

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

{

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

{

Point2D dungeonSpace(x, y);

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

return true;

}

}

return false;

}

**Entity.h**

#pragma once

#include "Point2D.h"

class Entity

{

protected:

Point2D\* mPosition;

public:

Entity();

~Entity();

void SetPosition(Point2D\* pos);

Point2D GetPosition();

};

**Entity.cpp**

#include "Entity.h"

Entity::Entity()

{

mPosition = new Point2D();

}

Entity::~Entity()

{

}

void Entity::SetPosition(Point2D\* pos)

{

mPosition = pos;

}

Point2D Entity::GetPosition()

{

return \*mPosition;

}

**Game.h**

#pragma once

#include "Item.h"

#include "Wumpus.h"

#include "Dungeon.h"

class Game

{

private:

Player \*mPlayer;

Wumpus \*mWumpus;

Dungeon \*mDungeon;

Item\* mTraps;

Item \*mGold;

public:

Game();

~Game();

void TitleScreen();

void PlayerMove();

bool HazardCheck();

bool PlayAgain();

};

**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(4,0));

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

mDungeon->GenRooms();

}

Game::~Game()

{

}

void Game::TitleScreen()

{

std::cout << "WUMPUS WORLD\n";

std::cout << "To go up type w or north. To go down type s or south.\nTo go left type west or a. To go right type east or d.\n";

system("pause");

system("CLS");

std::cout << "You take your first steps into the infamous Wumpus' swamp.\n";

std::cout <<"You've only heard of the fearsome beast in ancient tales told by your elders \nand now you stand firm in the beast's homeland.\n";

std::cout <<"Can you dodge the fearsome beast and his dastardly traps to find his\nillustrious gold? Find out... now...\n";

system("pause");

system("CLS");

}

void Game::PlayerMove()

{

String direction;

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

std::cin >> direction;

mPlayer->Move(direction);

system("pause");

system("CLS");

}

bool Game::HazardCheck()

{

bool event = false;

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

{

std::cout << "You stepped in the poison swamp! You feel the flesh melt from your bones until\nyou become a pile of mush.";

std::cout <<"Your finals moments are spent wondering how you got\nthrough the poison swamp on your way here... Game Over...\n";

return false;

}

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

{

std::cout << "The Wumpus found you! Before you can react the hideious monster delivers a\nmighty punch to your chest at an unimaginable speed!\n";

std::cout << "The punch breaks your ribs and leaves you gasping for air. The Wumpus surveys\nhis freshly caught prey and drags you deeper into the swamp... Game Over...\n";

return false;

}

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

{

std::cout << "You fell into a pit! Due to your now broken bones you are unable to climb out\nof the pit. ";

std::cout << "All you can do know is wait for the Wumpus to show up or starve\nto death... Game Over...\n";

return false;

}

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

{

std::cout << "You found the gold!\nYou can finally pay off your college debt!\n";

return false;

}

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

{

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

event = true;

}

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

{

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

event = true;

}

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

{

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

event = true;

}

if (mDungeon->DungeonBorderWarning(mPlayer))

event = true;

if (event == false)

{

std::cout << "You're standing in an empty field.\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.h**

#pragma once

#include "Entity.h"

class Item : public Entity

{

public:

Item();

~Item();

Item(Point2D \*pos);

Point2D GetPosition();

bool CheckForPlayer(Entity\* entity);

bool IsPlayerNearby(Entity\* entity);

};

**Item.cpp**

#include "Item.h"

Item::Item()

{

mPosition = new Point2D(4, 4);

}

Item::~Item()

{

}

Item::Item(Point2D \*pos)

{

mPosition = pos;

}

Point2D Item::GetPosition()

{

return \*mPosition;

}

bool Item::CheckForPlayer(Entity \* entity)

{

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

{

return true;

}

return false;

}

bool Item::IsPlayerNearby(Entity \* entity)

{

//Checks the positions above and below the Entity for the player

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

{

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

return true;

}

//Checks the positions to the left and right of the Entity for the player

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

{

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

return true;

}

return false;

}

**Player.h**

#pragma once

#include "Entity.h"

class Player : public Entity

{

public:

Player();

~Player();

void Move(String direction);

void SetPosition(Point2D\* pos);

Point2D GetPosition();

};

**Player.cpp**

#include "Player.h"

Player::Player()

{

mPosition = new Point2D(4,2);

}

Player::~Player()

{

}

void Player::Move(String direction)

{

if (String("north") == direction.LowerString() || String("w") == direction.LowerString())

{

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

std::cout << "You head north.\n";

}

else if (String("south") == direction.LowerString() || String("s") == direction.LowerString())

{

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

std::cout << "You head south.\n";

}

else if (String("west") == direction.LowerString() || String("a") == direction.LowerString())

{

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

std::cout << "You head west.\n";

}

else if (String("east") == direction.LowerString() || String("d") == direction.LowerString())

{

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

std::cout << "You head east.\n";

}

else

std::cout << "Invalid command.\n";

}

void Player::SetPosition(Point2D \* pos)

{

Entity::SetPosition(pos);

}

Point2D Player::GetPosition()

{

return \*mPosition;

}

**Point2D.h**

#pragma once

#include <iostream>

class Point2D

{

private:

int mX;

int mY;

public:

Point2D();

~Point2D();

Point2D(int x, int y);

Point2D operator+(const Point2D & other);

Point2D operator-(const Point2D & other);

Point2D operator+=(const Point2D & other);

Point2D operator-=(const Point2D & other);

bool operator==(const Point2D & other);

int GetX();

int GetY();

};

**Point2D.cpp**

#include "Point2D.h"

Point2D::Point2D()

{

mX = 1;

mY = 1;

}

Point2D::~Point2D()

{

}

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)

{

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;

}

**Wumpus.h**

#pragma once

#include "Player.h"

class Wumpus : public Entity

{

public:

Wumpus();

~Wumpus();

void SetPosition(Point2D\* pos);

Point2D GetPosition();

bool CheckForPlayer(Player\* player);

bool IsPlayerNearby(Player\* player);

};

**Wumpus.cpp**

#include "Wumpus.h"

#include "Point2D.h"

Wumpus::Wumpus()

{

mPosition = new Point2D(2, 1);

}

Wumpus::~Wumpus()

{

}

void Wumpus::SetPosition(Point2D \* pos)

{

Entity::SetPosition(pos);

}

Point2D Wumpus::GetPosition()

{

return \*mPosition;

}

bool Wumpus::CheckForPlayer(Player\* player)

{

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

{

return true;

}

return false;

}

bool Wumpus::IsPlayerNearby(Player\* player)

{

//Checks the positions above and below the Entity for the player

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

{

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

{

return true;

}

}

//Checks the positions to the left and right of the Entity for the player

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

{

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

{

return true;

}

}

return false;

}

**Main.cpp**

#include "Game.h"

#include <stdlib.h>

int main()

{

Game\* newGame = new Game();

bool gameRunning = true;

newGame->TitleScreen();

while (gameRunning)

{

newGame->PlayerMove();

if (!newGame->HazardCheck())

{

if (newGame->PlayAgain())

{

delete newGame;

newGame = new Game();

system("CLS");

}

else

gameRunning = false;

}

}

delete newGame;

}

**4: Read Me**

1. Start up your computer.
2. Sign into an account
3. Open up a web browser
4. Copy and paste: <https://github.com/ZacharyBriggs/AssesmentOne/releases/tag/1.0W> into the search bar of the browser.
5. Click on Wumpus\_World.zip
6. Find where the zip file was downloaded to.
7. Extract the files.
8. Read the read\_me for controls.
9. Run Wumpus\_World.exe
10. Have fun!