>>Introduction

Link to an unlisted gameplay video: https://youtu.be/bevBFVyRnCI

As part of the Ubisoft next competition I decided to challenge myself and attempt to complete it as a way to test my skills. For about 72 hours I have attempted to work on this game that derives off of the old bomberman game. I initially did this challenge as I was recommended to attempt it as a way to gain an internship at Ubisoft as well as a way to test my programming skills. The biggest challenge of the competition would have to be the fact that until this point I had never programmed in C++ and I have the most experience with C# and Java. Which meant that throughout the competition I had to constantly go through the C++ documentation to be able to use my previous knowledge of programming to do so in the language.

The game concept I ended up making was meant to be a basic drill mining game which I've called Drillperson. The gameplay loop is that you as the player take control of a drilling machine and are attempting to drill through as many walls and rock as possible. As the player drills away the green walls which represent what areas the player can break, they gain points but as they play on there is a timer that decreases in each level and if the player ever lets the time run out then the game is over and their final score is presented.

>> Explanation of the Code

When initially coding the game I wanted to add a similar feature of the original Bomberman in which the player could only destroy the blocks in a 15X13 area. So I first had to set up a 2D matrix which would constitute each position in the game where the player or a wall could be at. I currently have it set up so that the map size is 15X15 and having each point be a struct which contains the pixel position for the x and the y as well as the current object in the position.

```
//Current position and what is in that position. The string is mostly used for making the map and checking if the player is colliding with something
struct Position {
    int x;
    int y;
    string currentObj;
};
```

```
//Sets up the map size.
int mapSizeX = 15;
int mapSizeY = 15;
//Size of the boxes that will be drawn for the walls and finish post
int boxSize = 0;

//Sets up the 2d matrix map for each level using the position struct
Position gameMap[15][15] = {};
```

The size of each level can be changed by affecting the map size variables near the top of the script.

I later set up the player movement so that they could only appear at one of the different positions that are saved in the 2D Matrix. This was done by checking the players current position which correlates with the x and y of the 2D array and then when the player pressed “W”, “A”, “S”, “D” then they would move forward one spot which would be to the next matrix position. The example code below is for moving the character towards the right. I would also create a basic box that would be rendered for the spots so that I could more easily determine if the player was aligning with the correct spots.

```
//Checks the strings in the 4 adjacent blocks to the character and saves them to a string variable
string rightPosObj = gameMap[player.x + 1][player.y].currentObj;
string leftPosObj = gameMap[player.x - 1][player.y].currentObj;
string upPosObj = gameMap[player.x][player.y + 1].currentObj;
string downPosObj = gameMap[player.x][player.y - 1].currentObj;
```

```
//If the D key was pressed on a windows divice and if the spot to the right isnt a wall then it will move the character one spot in the grid to the right
if (App::GetController().GetLeftThumbStickX() > 0.5f && rightPosObj != "Wall" && rightPosObj != "Breakable Wall")
{
    //timeButtonPressed += deltaTime;
    player.x = player.x + 1;
    player.lastMovement = "Right";
    int x = player.x;
    int y = player.y;
    //Grabs the map cordinates to place the player sprite
    int gameMapX = gameMap[x][y].x;
    int gameMapY = gameMap[x][y].y;
    //Updates the players sprite and animation
    testSprite->SetPosition(gameMapX + 25, gameMapY + 25);
    testSprite->SetAnimation(ANIM_RIGHT);
}
```

At this point I now needed to work on the walls so that the player could know where it could or could not go. This was done by repurposing the box code I had been using as a way to determine where the player should be, and then changing the color depending on the wall type and adding a couple of checks in the movement to make sure the player didn't just pass through them as if they didn't exist. I also implement a seeded random value using C++ srand() and rand so that it would randomly create a number at each position to determine if it should create a breakable wall or not. This allows some variation in each level so that there is some challenge and the player needs to think about how to break as much as possible before the time runs out.

```
for (int i = 0; i < mapSizeX; i++) {
    for (int j = 0; j < mapSizeY; j++) {
        int randomInt = rand(); //Randomly gives a number between 0 and 32767 while using the curre
        string currentPosObj = gameMap[i][j].currentObj;
        if (i == 0 || j == 0 || i == 14 || j == 14 || ((i%2 == 0) && (j%2 == 0))) {
            gameMap[i][j] = { i * 50 + 5, j * 50 + 5, "Wall" }; // initialize position with x and y
        }
        else if (i == 13 && j == 13) {
            gameMap[i][j] = { i * 50 + 7, j * 50 + 7, "End Point" }; // initialize position with x a
        }
        else if (randomInt < 10922 && currentPosObj != "Player" && currentPosObj != "End Point") {
            gameMap[i][j] = { i * 50 + 7, j * 50 + 7, "Breakable Wall" }; // initialize position wit
        }
        else {
            gameMap[i][j] = { i * 50 + 7, j * 50 + 7, "Empty" }; // initialize position with x and y
        }
    }
}
```

Breakable walls were being colored green and made of length 46 while the main walls were colored red with a length of 50. This size difference was so that the lines did not intercept and cover each other causing confusion to the player. I had also added an end state so the player knew where to go instead of aimlessly wandering around the map which is what the “End point” variable signified.

I then implemented a basic check to see if the player had recently pressed the button E and was pointing at a breakable wall or the end point. Then it would just replace the wall with an empty wall which doesn't get drawn or in the case of the end point just create a new level.

```
if (GetAsyncKeyState(0x45)) {
    if (player.lastMovement == "Right" && rightPosObj == "Breakable Wall") {
        gameMap[player.x + 1][player.y].currentObj = "Empty";
        IncreaseScore();
        testSprite->SetAnimation(ANIM_RIGHT);
    }
    else if (player.lastMovement == "Left" && leftPosObj == "Breakable Wall") {
        gameMap[player.x - 1][player.y].currentObj = "Empty";
        IncreaseScore();
        testSprite->SetAnimation(ANIM_LEFT);
    }
    else if (player.lastMovement == "Up" && upPosObj == "Breakable Wall") {
        gameMap[player.x][player.y + 1].currentObj = "Empty";
        IncreaseScore();
        testSprite->SetAnimation(ANIM_FORWARDS);
    }
    else if (player.lastMovement == "Down" && downPosObj == "Breakable Wall") {
        gameMap[player.x][player.y - 1].currentObj = "Empty";
        IncreaseScore();
        testSprite->SetAnimation(ANIM_BACKWARDS);
    }
    else if (gameMap[player.x][player.y].currentObj == "End Point") {
        //End of level game
        NewLevel();
    }
}
```

I finally decided to add some GUI and a scoring system so that the player could know how well they were doing and how to roughly play the game.

```
void GuIDisplay() {
    App::Print(765, 745, "Current Level:");
    App::Print(885, 745, std::to_string(currentLevel).c_str());
    App::Print(765, 720, "Current Score:");
    App::Print(890, 720, std::to_string(player.playerScore).c_str());

    float timeInFloat = timeLeftLevel / 60000;
    string timerInString = std::to_string(timeInFloat);
    App::Print(765, 695, "Time left:");
    App::Print(845, 695, timerInString.c_str());

    App::Print(765, 650, "Goal of game:");
    App::Print(765, 625, "Destroy as green blocks!");
    App::Print(765, 600, "Each green block destroyed");
    App::Print(765, 580, "is 50 points!");
    App::Print(765, 555, "Finish the level before timer");
    App::Print(765, 535, "runs out.");
    App::Print(765, 500, "Press W to move Up");
    App::Print(765, 475, "Press S to move Down");
    App::Print(765, 450, "Press A to move Left");
    App::Print(765, 425, "Press D to move Right");
    App::Print(765, 400, "Press E to activate action.");
}
```

Link to an unlisted gameplay video:
https://youtu.be/bevBFVyRnCI