Feature Documentation

The Player

The player is loaded in at spawn point based on the loaded level and from their the user can move using wasd and shoot using the arrow keys. The player can also collect keys and use these to open door.

Game.cpp

(Global Scope)

```cpp
        }
        dx = 0;
        dy = 0;

        if(game->keys['W'])
        {
            dy = 1.0f;
        }
        if(game->keys['A'])
        {
            dx = -1.0f;
        }
        if(game->keys['S'])
        {
            dy = -1.0f;
        }
        if(game->keys['D'])
        {
            dx = 1.0f;
        }
        if (game->keys['W'] || game->keys['A'] || game->keys['S'] || game->keys['D'])
        {

            if (game->player1.speed <= maxSpeed-accel)
            {
                game->player1.speed += accel;
            }
            else
            {
                game->player1.speed = maxSpeed;
            }

        }
        else
        {
            if (game->player1.speed >= accel)
            {
                game->player1.speed -= accel;
            }
            else
            {
                game->player1.speed = 0;
            }
```

91 %

This code fields wasd and changes the velocity the player will move when player.move is run

Game.cpp

(Global Scope)

```cpp

        updateGame(game, dx, dy, secsPassed);
        if (game->keys[VK_UP])
        {
            if(game->player1.curCoolDown ==0 )
            {
                addShot(game, game->player1.x, game->player1.y, game->shotTex, entity::UP);
                game->player1.curCoolDown = game->player1.shotCoolDown;
            }
            game->player1.facing = entity::DOWN;
        }
        else if (game->keys[VK_LEFT])
        {
            if(game->player1.curCoolDown ==0 )
            {
                addShot(game, game->player1.x, game->player1.y, game->shotTex, entity::LEFT);
                game->player1.curCoolDown = game->player1.shotCoolDown;
            }
            game->player1.facing = entity::LEFT;

        }
        else if (game->keys[VK_RIGHT])
        {
            if(game->player1.curCoolDown ==0 )
            {
                addShot(game, game->player1.x, game->player1.y, game->shotTex, entity::RIGHT);
                game->player1.curCoolDown = game->player1.shotCoolDown;
            }
            game->player1.facing = entity::RIGHT;
        }
        else if (game->keys[VK_DOWN])
        {
            if(game->player1.curCoolDown ==0 )
            {
            addShot(game, game->player1.x, game->player1.y, game->shotTex, entity::DOWN);
            game->player1.curCoolDown = game->player1.shotCoolDown;
            }
            game->player1.facing = entity::UP;
        }
```

Fields the arrow keys and spawns a shot entity at the player facing in the direction of the arrow key. Shots move based on their initial direction and cannot change. There is a cooldown for shots so users have limited fire power

Game.cpp

(Global Scope)

```cpp
void updateGame(Game* g , double dx, double dy, long double time)
{
    time *= 20;
    if(game->player1.speed != 0)
    {
        game->player1.move(game, dx, dy, time);
    }

    enemy* enemytest = g->enemies;
    for(int i = 0; i < g->enemyArraySize; i++)
    {
        if(enemytest->isValid)
        {
            enemytest->move(g, g->player1.x - enemytest->x, g->player1.y - enemytest->y, time);
        }
        enemytest++;

    }

    shot* shotTemp = g->shots;
    for(int i = 0; i < g->shotArraySize; i++)
    {
        if(shotTemp->isValid)
        {
            shotTemp->move(g, time);
        }
        shotTemp++;
    }



    if(g->player1.curCoolDown < 0)
    {
        g->player1.curCoolDown = 0;
    }
    if(g->player1.curCoolDown > 0)
    {
        g->player1.curCoolDown -= time;
    }

}
```

In update game we move the player, enemies and shots as well as cool down the players shot timer. In the game removed enemies are zeroed in the game array so the boolean isvalid will be false. This means only entities that have been initialised are rendered and collided.

Animation

Both enemies and the player are animated using 8 images(1 for each animation) that cycle when they move. The current frame is stored in their parent entity.

Game.h Game.cpp

entity

```
class entity
{
public:
    int facing;
    const static int UP = 0;
    const static int LEFT = 3;
    const static int DOWN = 2;
    const static int RIGHT = 1;

    double x;
    double y;
    double height;
    double width;
    int texID;
    double curAnimFrame;
```

Game.h Game.cpp

player

```

    if(xMoveGood || yMoveGood)
    {
        curAnimFrame += speed*time/4;
    }
    if(curAnimFrame > 8)
    {
        curAnimFrame = (int)curAnimFrame % 8;
    }
```

Right:
Declaration of current Animation frame
Above:
Update of current Animation frame. Called in player.move(shown above) and enemy.move

Game.h Game.cpp

(Global Scope)

```


        //draw enemies(push and pop per enemy)
        for(int i = 0; i <= g->enemyArraySize; i++)
        {
            glPushMatrix();
                drawing = &g->enemies[i];
                if(&g->enemies[i] != 0)
                {
                    glTranslatef(drawing->x, drawing->y, 0);
                    glRotatef(90 * drawing->facing, 0, 0, 1);
                    glBindTexture(GL_TEXTURE_2D, g->enemyTex[(int)drawing->curAnimFrame]);
                    fillRectangleFromCentre(drawing->width, drawing->height);
                }
            glPopMatrix();
        }
```

when drawing enemies (and player) the texture that is used is gotten from an array of texture handles using the truncated value of the current animation counter.

Game.h Game.cpp

(Global Scope)

```cpp

void initGame(Game* g)
{
    g->floorTex = loadPNG("metal_plates.png");
    g->wallTex = loadPNG("wall.png");
    g->shotTex = loadPNG("shot.png");
    g->playerTex[0] = loadPNG("player1.png");
    g->playerTex[1] = loadPNG("player2.png");
    g->playerTex[2] = loadPNG("player3.png");
    g->playerTex[3] = loadPNG("player4.png");
    g->playerTex[4] = loadPNG("player5.png");
    g->playerTex[5] = loadPNG("player6.png");
    g->playerTex[6] = loadPNG("player7.png");
    g->playerTex[7] = loadPNG("player8.png");
    g->enemyTex[0] = loadPNG("enemy1.png");
    g->enemyTex[1] = loadPNG("enemy2.png");
    g->enemyTex[2] = loadPNG("enemy3.png");
    g->enemyTex[3] = loadPNG("enemy4.png");
    g->enemyTex[4] = loadPNG("enemy5.png");
    g->enemyTex[5] = loadPNG("enemy6.png");
    g->enemyTex[6] = loadPNG("enemy7.png");
    g->enemyTex[7] = loadPNG("enemy8.png");
    g->doorTex = loadPNG("door.png");
    g->stairTex = loadPNG("stairs.png");
    g->keyTex = loadPNG("key.png");

```

Textures are loaded in the init game function which is called before the start of the game loop. Static textures are stored as int and textures for animated entities are stoered in an int array.

Collision

Collision is done using a collection of collide functions for each different type of object in the game(map, player, shots, enemies, doors and stairs).

Game.h × Game.cpp

entity

```
door* collideDoors(Game* g, double x, double y, double w, double h);
bool collideStairs(Game* g, double x, double y, double w, double h);
enemy* collideEnemies(Game* g, double x, double y, double w, double h);
bool collideShots(Game* g, double x, double y, double w, double h);
bool collideMap(Game* g, double x, double y, double w, double h);
bool collidePlayer(Game* g, double x, double y, double w, double h);
```

Game.h × Game.cpp

entity

```

struct Game
{
    int floorTex;
    int wallTex;
    int shotTex;
    int playerTex[8];
    int enemyTex[8];
    int doorTex;
    int stairTex;
    int keyTex;
    bool done;
    bool keys[256];
    double mouseXPos;
    double mouseYPos;

    Tilemap tileMaps[16];
    Tilemap* curTileMap;
    int levelCount;
    int curLevel;

    stair stairs;

    door doors[200];
    const static int doorArraySize = 200;

    player player1;

    enemy enemies[200];
    const static int enemyArraySize = 200;

    key gameKeys[200];
    const static int keyArraySize = 200;

    shot shots[200];
    const static int shotArraySize = 200;



};
```

Collision with the map is done by integer dividing the passed x, y and x+w and y + h by the tilesize(16) and using the value to access the tile this x and y sit on. If the tile is denoted as filled the function returns true;

```
Game.h    Game.cpp
(Global Scope)
bool collideMap(Game* g, double x, double y, double w, double h)
{
    return !((!g->curTileMap->tiles[(int)(y - h / 2) / TILESIDE][(int)(x - w / 2) / TILESIDE].filled) &&
        (!g->curTileMap->tiles[(int)(y + h / 2) / TILESIDE][(int)(x + w / 2) / TILESIDE].filled) &&
        (!g->curTileMap->tiles[(int)(y + h / 2) / TILESIDE][(int)(x - w / 2) / TILESIDE].filled) &&
            (!g->curTileMap->tiles[(int)(y - h / 2) / TILESIDE][(int)(x + w / 2) / TILESIDE].filled));
}
```

Collision with everything else is done using axis aligned bound boxes. Each collision method is largely the same but for instance if its enemies or keys it loops through all them and checks each but for player or stairs it doesn't need to loop as only one exists.

Game.h Game.cpp

(Global Scope)

```
bool collidePlayer(Game* g, double x, double y, double w, double h)
{
    player* test = &g->player1;


    if(test->isValid)
    {
        if (!(y > test->y + test->height ||
            y + h < test->y ||
            x > test->x + test->width ||
            x + w < test->x))
        {
            return true;
        }

    }
    return 0 ;
}
key* collideKeys(Game* g, double x, double y, double w, double h)
{
    key* test = g->gameKeys;
    for (int i = 0; i < g->keyArraySize; i++)
    {
        if(!(x == test->x && y == test->y))
        {
            if(test->isValid)
            {
                if (!(y > test->y + test->height ||
                    y + h < test->y ||
                    x > test->x + test->width ||
                    x + w < test->x))
                {
                    return test;
                }
            }
        }
        test++;
    }
    return 0;
}
```

Acceleration

The players movement in the game is not instantaneous and despite it not taking long his speed does increase at the start of his movement.

Game.h Game.cpp

(Global Scope)

```
        if (game->keys['W'] || game->keys['A'] || game->keys['S'] || game->keys['D'])
        {

            if (game->player1.speed <= maxSpeed-accel)
            {
                game->player1.speed += accel;
            }
            else
            {
                game->player1.speed = maxSpeed;
            }

        }
        else
        {
            if (game->player1.speed >= accel)
            {
                game->player1.speed -= accel;
            }
            else
            {
                game->player1.speed = 0;
            }

        }
```

Game.h Game.cpp

player

```
void player::move(Game* g, double dx, double dy,  double time)
{
    bool yMoveGood = false;
    bool xMoveGood = false;
    double newX;
    double newY;
    double length = sqrt(dx*dx + dy*dy);
    if(length){
        newX = x + (dx*speed*time/length);
        newY = y + (dy*speed*time/length);
    }
    else
    {
        newX = x;
        newY = y;
    }
```

LevelSets and TileMaps

To make it easy to store levels in the filesystem each level is stored as a csv file using integers to denote what should be done for that tile(0 = floor, 1 = wall, 2 = floor + spawn enemy, 3 = floor +

spawn player, 4 = floor + spawn door and 5 = floor + spawn stairs).

Level3.txt - Notepad

File Edit Format View Help

```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1
1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1
1,0,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1
1,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,1,0,1,0,0,0,0,0,1,0,1
1,0,1,1,0,1,1,0,1,0,1,0,0,1,0,1,1,1,0,1,1,0,1,0,0,0,0,1,0,1,0,1
1,0,1,0,0,0,0,0,1,0,1,1,0,1,0,1,2,1,0,1,0,0,1,1,1,1,1,1,0,1,0,1
1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,1
1,0,1,1,1,1,1,0,1,1,1,1,0,1,0,1,0,0,0,1,1,1,0,1,0,0,0,0,0,1,0,1
1,0,1,0,0,4,1,3,1,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,1
1,0,1,0,1,4,1,1,1,1,0,1,1,1,1,1,0,1,0,1,0,0,0,1,1,1,1,1,1,1,0,1
1,0,1,0,1,4,0,5,1,0,0,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1
1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1
1,0,1,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,2,2,2,1,0,0,0,1,1,1,0,1
1,0,1,0,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,2,1,0,1
1,0,1,0,1,0,1,1,1,0,1,0,0,0,2,0,0,0,0,0,0,0,2,1,0,0,0,1,0,1,0,1
1,0,1,0,0,0,1,2,1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,0,1
1,0,1,0,1,0,1,0,1,0,1,2,1,0,0,0,0,0,0,0,0,0,0,1,0,1,2,2,0,1,0,1
1,0,1,0,1,0,1,0,1,0,1,1,1,0,0,0,0,0,0,1,0,0,0,1,0,1,1,1,0,0,0,1
1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,1
1,0,1,2,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,1,1,1,1,1,0,1
1,0,1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1
1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,0,1,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,1
1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,1,0,1,1,1,1,0,1
1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,2,0,0,0,0,2,1,0,1
1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,2,0,0,0,0,2,1,0,1
1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,2,0,0,0,0,2,1,0,1
1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

Levels are loaded into the game using the order found in levels.txt. The program to both find the tilemaps and work out the level order in the game. Changing the order in this file changes the order and you can als add and remove levels by editing this file.

levels.txt - Notepad

File Edit Format View Help

```
Level1.txt
Level2.txt
Level3.txt
```

Level Editor

The level editor is copied from the source of the game with a few changes. The ai and movement code as been commented out or deleted and mouse controls added. Clicking on a tile increments its tile id mod 6 and then reloads the tilemap. When the user launches the editor they are presented with a tile map which is four walls and floor with nothing in it. Once the user has set up their tilemap they hit space and it writes the tile ids to a file called newLevel.txt which they can rename and add to levels.txt as explained before.

This game.cpp id editor/game.cpp where all above are Graphicsgame/game.cpp. Ill note whenever it is editor/game.cpp

Game.cpp

(Global Scope)

```cpp

        case WM_LBUTTONDOWN:
            {
                SetCursor(NULL);
                cycleTile(game, game->mouseXPos, game->mouseYPos);
            }
        break;

        case WM_LBUTTONUP:
            {
            }
        break;

        case WM_MOUSEMOVE:
            {

                game->mouseXPos = (LOWORD(lParam) - windowWidth/2)/camZoom + camX;
                game->mouseYPos = ((windowHeight - HIWORD(lParam)) - windowHeight/2)/camZoom + camY;
            }
```

Mouse Controls

```cpp
void cycleTile(Game* g, float x, float y)
{
    Tile* t;
    if(!(x < 0 || y < 0))
    {
        t = &g->curTileMap->tiles[(int)y/TILESIDE][(int)x/TILESIDE];
        t->tileID++;
        if(t->tileID == 6)
            t->tileID = 0;
        refreshTileMap(g);
    }

}
```

change tile id based on the ingame location of the users click

Time Stepping

Game.cpp

(Global Scope)

```cpp

    QueryPerformanceFrequency(&cps);
    double secsPassed = 0;
    QueryPerformanceCounter(&curCount);
    while(!game->done)
    {
        prevCount = curCount;
        QueryPerformanceCounter(&curCount);
        countDifference = curCount.QuadPart - prevCount.QuadPart;
        secsPassed = (long double)countDifference / (long double)cps.QuadPart;
```

The seconds passed between frames is calculated at the start of each loop and used in the move functions of all the entities

Game.h Game.cpp

player

```cpp
void player::move(Game* g, double dx, double dy,  double time)
{
    bool yMoveGood = false;
    bool xMoveGood = false;
    double newX;
    double newY;
    double length = sqrt(dx*dx + dy*dy);
    if(length){
        newX = x + (dx*speed*time/length);
        newY = y + (dy*speed*time/length);
    }
    else
    {
        newX = x;
        newY = y;
    }

```

Scoring

If a player finishes the game their score is the total number of keys they collected. This score ignores keys spent on doors. This score is presented upon reaching the end of the final level.

Game.h Game.cpp

player

```
if(key* k = collideKeys(g, x, y, width, height))
{
    keyCount++;
    score++;
    removeKey(g, k);
}
```

Add 1 to your score

Game.h Game.cpp

player

```
        curAnimFrame = (int)curAnimFrame % 8;
    }
    if(collideStairs(g, x, y, width, height))
    {


        if(g->curLevel == g->levelCount)
        {
            char* winmsg = "";
            sprintf(winmsg, "you win. you collected  %d keys. well done", score);
            MessageBox(NULL, winmsg, "WINNNNNNERNERENRNERNERN", MB_OK);
            g->done = true;
        }
        else
        {
            setWorld(g, g->curLevel + 1);
        }
    }
```

Display score when game finishes

User Manual

Launching the Game

To start playing with the starting levels run GraphicsGame.exe

Controls are wasd to move and the arrow keys to shoot in the four directions.

The aim of the game is to reach the stairs in each level and by reaching them in the last level win the game.

Stairs

if an enemy touches you you lose the game and by shooting enemies they will drop keys that you can pick up to open doors(each door requires 5 keys).

Door Key

Creating levels

You can add your own levels to the game by running Editor.exe. Once it is open you can click on tiles to cycle to change what they are (floor, wall, enemy, player, door, stairs). NOTE: a level can only have one player and stairs.

Controls are wasd to pan around and shift to pan quicker, e and q can be used to zoom in and out. Press space when your happy and it will generate a file called newLevel. This is your level and to add it to the game you need to rename it to what you want it to be called and add it as a new line to the file called levels. Levels takes the top of the list as level one and works down so plan your level difficulty accordingly. Feel free to remove the default levels and create a set of levels for you and others to try.

Kyle Alexander 100082709