

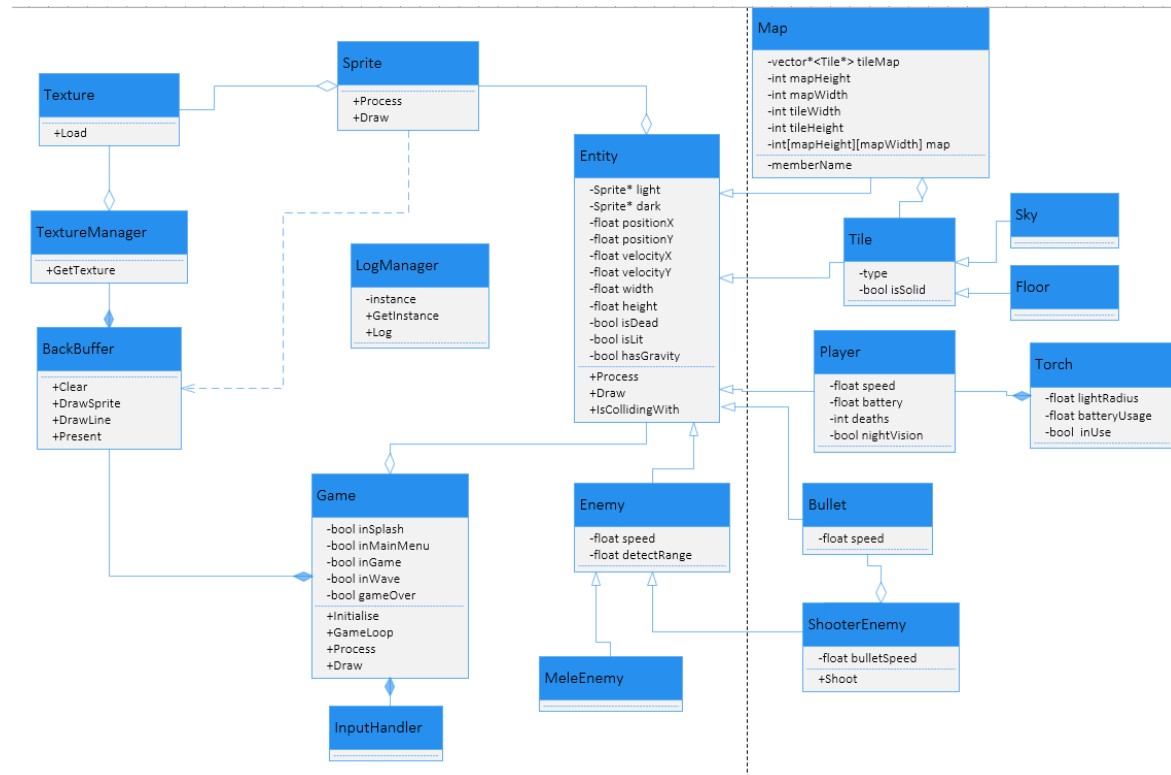
25/9:

The team decided to work on a platforming game, we are looking a ways to make it unique and fun for the player. I have been assigned to work on the UML diagram for the game. I started work on it have included some basic interactions between the core game engine and the player.

2/10:

The team further discussed how to make our game unique. We are going to implement a Torch function into the game, the player will have to manage a battery level, while navigating a map of hidden dangers. The torch will toggle with leftclick and change the sprites (brighter) of objects around the mouse. Some objects such as traps or enemies may only be visible in either light or darkness.

I finished the UML Diagram I was mainly focusing on how the torch will be interacting with other entities and how it will be coded. The rest of the games architecture is pretty standard.



Next I created the control scheme for the game. We will not be including Xbox controller compatibility. The game will be using standard platformer movement controls, with Spacebar for Jump and W/S for climbing. The torch will be toggled with the left mouse button.



Right	D
Left	A
Jump	SPACE
Up (ladders/rope)	W
Down (ladders/rope)/Drop down	S
Torch/Light	Left-click (hold)
Interact/Use	E
Pause Menu/Settings	ESC



I read over the current versions of the GDD and TDD to make sure my understanding of the game is correct. I expanded the GDD in places I could add value. This included more in-depth descriptions of the player, torch, and enemy objects. (expansions in red)

Key Technical Challenges: Animation, level design. Possible solutions are using premade animated sprites and map tile sets.

Most of us are not familiar with working with sprites and animation so we believe that animation would be a more difficult area of the games development. Especially with one of our core features being a flashlight, that will alter the visibility, sprites, or animations of the game's entities. |

Key Algorithms:

Enemy States: The games enemies will use a collision-based detection to switch from a passive state into an aggressive state. The enemy will switch back into a passive state after the enemy leaves the detection range (possible timer to maintain the chase state for a short duration). The aggressive and passive states will be handled differently based on the enemy attack being ranged or melee.

Flashlight: The flashlight is one of the games key features. The game will naturally be a dark theme, with limited visibility. The entities in the game will toggle between lit and unlit states. When the light is colliding with an entity, it will switch its state. The ground/scenery will simply change its lighting. While enemies will toggle between visible and invisible to the player. We currently plan for 2 main enemy types. Solid enemies will be invisible to the player while in the dark and become visible when the light is on them. Ghost type enemies will be visible in the dark and disappear when the light is shone on them.

Mele Enemy States: Mele enemies will either walk back and forward or stand idle while in their passive state. When they are in the chase state, they will path horizontally towards the players Xposition. They will be affected by gravity, and so naturally fall downwards if they walk off the edge of a platform. If the enemies collide with a wall while chasing the player, they will consistently jump, either following the player, or jumping next to the wall until the player leaves their detection range, or the player passes them, and they start moving in the other direction.

Ranged Enemy States: Ranged enemies will stand idle in their passive state. While in their aggressive state they will continuously shoot projectiles towards the player based on a timer. There is potential for multiple types of ranged enemies, adjusting the projectiles gravity, speed, directional vs aimed projectiles.

Gravity: As a platformer, the gravity is a key aspect of the game. The games entities will have a gravity variable, initialised based on the entity type. The gravity is applied as a downwards velocity every frame.

9/10:

After creating the UML diagram I had a decent understanding of how the games classes would interact. I read over the key algorithms in the TDD and expanded on them with my understanding of how the games features will be implemented.

13/10

I Expanded GDD core gameplay and we assigned tasks. I'm coding the torch, Maaka is doing the player, Adriaan is doing the map, and Edward is making the enemies.

Dealing with enemies:

Enemies will consist on one or two types depending on the time schedule, enemies are yet to be given a theme but one will run at the player to melee attack, while another enemy type will have a ranged attack, which can be dodged by the player. These enemies may or may not be visible based on the light level. The flashlight, when on will freeze the enemy if it shines upon it.

Flashlight:

The flashlight will turn on when the player holds left click, and will light up an area around the mouse position. This will be handled by detecting collisions around the mouse within the light radius, objects hit by the light will toggle a lit/unlit state. Most objects will simply change sprite based on the lit/unlit state, however there can be some enemies/traps that are invisible to the player in the dark and require the light to see, and some can work the opposite way. The flashlight will have a battery level that depletes with use, this will either refill passively overtime, or with collectable items.

Gravity:

Gravity will be a global value applying downward velocity to all entities based on whether Gravity is on/off for each entity. This will be an easily changable value based on gameplay testing.

14/10

I wrote some psuedo code for the basic torch interactions. I listed out the variables the torch would need to function properly, and everywhere it would interact with other classes. It's a lot easier to visualize how it fits into the game when its all on one page, without any irrelevant code.

```
Torch:
    Sprite = crosshair
    bool isOn
    float batteryLevel
    float depleteRate
    float restoreRate
    float radius

    Process()
        moveToMouse

        if(isOn)
            batteryLevel - depleteRate
        else
            batteryLevel + restoreRate

InputHandler:
    leftclick Down
        game.torch->setOn(true)

    leftclick Up
        game.torch->setOff(true)

Game:
    Process()
        Torch->Process()
        checkTorchLight()

    CheckTorchLight()
        for(entityList)
            if(((entityX-TorchX)*(entityX-TorchX) + (entityY-TorchY)*(entityY-TorchY)) < r*r);
                entity->setIsLit(true)
            else
                entity->setIsLit(false)

    Draw()
        drawTorch

Entity:
    Process()
        if(isLit)
            initialise(lightSprite)
        else
            intiialise(darkSprite)
```

15/10

I started coding the core of the flashlight functionality at home before class, unfortunately my internet shut off and I wasn't able to finish it. I went into Uni early to work on it from there, I was able to get the main flashlight function working, the blocks in a set radius around the flashlight switch to a light version of the sprite.

```
void Torch::Process(float deltaTime)
{
    // Keep Torch on Mouse
    int mouseX;
    int mouseY;
    SDL_GetGlobalMouseState(&mouseX, &mouseY);

    SetPositionX(mouseX);
    SetPositionY(mouseY);
}
```

```
void Game::CheckTorchLight()
{
    for (size_t i = 1; i < tileMap.size(); i++)
    {
        float entityX = tileMap.at(i)->GetPositionX();
        float entityY = tileMap.at(i)->GetPositionY();

        float torchX = torch->GetPositionX();
        float torchY = torch->GetPositionY();
        float radius = torch->GetRadius();

        if (((entityX - torchX)*(entityX - torchX) + (entityY - torchY)*(entityY - torchY)) < radius*radius)
        {
            if (torch->GetTorchOn())
            {
                tileMap.at(i)->SetIsLit(true);
                tileMap.at(i)->Initialise(tileLight);
            }
            else
            {
                tileMap.at(i)->SetIsLit(false);
                tileMap.at(i)->Initialise(tileDark);
            }
        }
        else
        {
            tileMap.at(i)->SetIsLit(false);
            tileMap.at(i)->Initialise(tileDark);
        }
    }
}
```

The torch currently loops through the tileMap as they are the only other completed object in the game at the moment. Initially I was checking if the torch was on before looping to try be more efficient, however this meant when the torch went off the sprites weren't switch back to their dark sprite. It needs to check the collisions even when the torch is off, so it resets the everything properly.

After work that night my internet was still off so I couldn't work on it from home. I still have to toggle the flashlight on/off with left mouse button, and set it up to use a battery level.

16/10

There was still no internet this morning, I went into uni early to continue working on the torch. I was able to get it to function smoothly with Left click to turn on and off, and a battery level that depletes while its on, and restores while its off. These values are all setup to be easily changed for balance testing.

```
void Torch::Process(float deltaTime)
{
    // Keep Torch on Mouse
    int mouseX;
    int mouseY;
    SDL_GetGlobalMouseState(&mouseX, &mouseY);

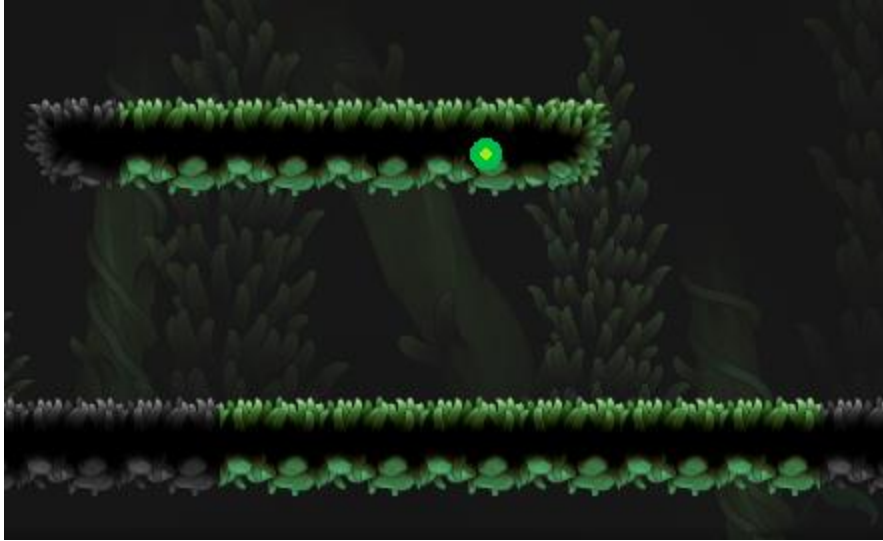
    SetPositionX(mouseX);
    SetPositionY(mouseY);

    // Manage Battery
    if (GetTorchOn())
    {
        SetBatteryLevel(GetBatteryLevel() - (depleteRate * deltaTime));
    }
    else
    {
        SetBatteryLevel(GetBatteryLevel() + (restoreRate * deltaTime));
    }
}
```

The only issue I faced was that when the torch runs out of battery, if the player keeps holding leftclick it will rapidly flash on and off. It runs out of battery, instantly restores 1 and turns back on, just to run out against straight away. I fixed this by adding a new control boolean. The input handler sets TorchOn to true, and the process method sets the torch state based on the TorchOn variable. When the torch runs out of battery, TorchOn is set to false.. The player has to reclick again to turn the light back on.

```
void Game::ManageTorch(float deltaTime)
{
    // Check Torch Battery Level
    if (torch->GetBatteryLevel() < 0)
    {
        torchOn = false;
        PlayTorchSound();
    }

    // Toggle Torch Based on Input Variable
    if (torchOn)
    {
        torch->SetTorchOn(true);
    }
    else
    {
        torch->SetTorchOn(false);
    }
}
```





In class today we presented a delta demo of the game. Currently the player movement isn't working correctly, and the enemies are not implemented yet. The torch was working properly, and the map also worked, but it used hard coded coordinates to place the blocks around. It is not efficient when we are using hundreds of tiles each level, and creating each level will require manual input of each tile's position. This makes it harder to expand the game so I gave some recommendations on using an Int array to loop and place tiles on the tileMap.


```
tileMap[15]->SetPosition(10, 800);  
tileMap[16]->SetPosition(60, 800);  
tileMap[17]->SetPosition(110, 750);  
tileMap[18]->SetPosition(160, 750);  
tileMap[19]->SetPosition(210, 700);  
tileMap[20]->SetPosition(260, 700);
```



17/10


My teammate handling the map was stuck on how to implement a system to build the levels. I sent him the source code I used to handle the map in my individual game, with the irrelevant parts erased. Instead of hard coding the coordinates of each tile, this system uses an array of integers, that are looped through. The position in the array matches to a position on the tile map, and the coordinates are set automatically.


**KKieraNN** 10/17/2020


 **Map.cpp**
1.39 KB



 **Map.h**
988 bytes




 **game.cpp**
2.90 KB



https://youtu.be/leaxE_waDNc

I basically followed this guys guide and adapted it to fit my game
If you set the tile width and height, just loop over the arrays and it adds tiles to the list with their positions automatically set
I initialized the tiles in the game.cpp but you can do it in the map.cpp as well
I was just working off what I already had and it was easier for me that way
My enemies walked along a path so I had to handle a lot more shit in the map class
I will try have FMOD and TTF for sound and hud setup tomorrow night

**Pancake** 10/17/2020
alright thanks i'll give it a go

```
int map[9][16] =
{
    { 0, 0, 0, 4,10,10,10,10,10, 5, 0, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 8, 3, 3, 3, 3, 3, 9, 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, 0,32,11,11,11,11,11,33, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0,11, 0, 0, 0, 0, 0,11, 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0,11, 6, 7, 0, 0, 0,11, 0, 0, 0, 0, 0 },
    { 0, 0, 1, 0,11, 4, 5, 0, 0, 0,11, 0, 0, 0, 0, 0 },
    { 30,11,11,11,31, 4, 5, 0, 0, 0,34,11,11,11,11,35 },
    { 0, 0, 0, 0, 0, 4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
};
```

```

for (int row = 0; row < 9; row++)
{
    for (int column = 0; column < 16; column++)
    {
        type = map[row][column];
        posX = column * GetTileWidth();
        posY = row * GetTileHeight();

        if (type < 30)
        {
            switch (type)
            {
                // Tiles
                case 0: // Snow tile
                    t = new Tile(posX, posY, type);
                    t->SetHasBlock(false);
                    t->SetBlock(NULL);
                    break;
                case 11: // Path tile
                    t = new Tile(posX, posY, type);
                    t->SetHasBlock(false);
                    t->SetBlock(NULL);
                    break;
            }
        }
    }
}

```

These are sections of code from my map.cpp and map.h I used in my for my individual game. I had to handle a lot more tiles as my enemies moved along a set path.

20/10

I updated the flashlight so that it will work with all the entities in the game. I started by writing psuedo code for everywhere the torch will have to interact with the rest of the game.

```
Game:
    vector EntityList

Entity:
    bool isLightable
    Sprite* CurrentSprite
    Sprite* SpriteLight
    Sprite* SpriteDark

    Initialise()
        set CurrentSprite (sprite)
        set SpriteLight (sprite)
        set SpriteDark (sprite)

        setIsLit(false)
        setLightable(false)

    SetSprites(spriteLight SpriteDark)
        set SpriteLight (spriteLight)
        set SpriteDark (SpriteDark)

        setLightable(true)

    ChangeSprite(sprite newSprite)
        CurrentSprite = newSprite

Game:
    CheckTorchLight()
        for EntityList
            if isLightable
                CheckCollisions

                SetIsLit
                entity->at(i) -> ChangeSprite(entity->at(i) -> light/dark)
```

My plan is for every entity in the game to hold 3 sprites. The default entity initialise will set all of these to the same sprite, and set isLightable to false. This is incase someone accidentally adds an entity to the torchList (renamed entity list). Using SetSprites(light, dark) will set the light and dark sprites, and set isLightable to true. This sets up the entity so that the torch can handle it properly, after that it can be added to the torchList with torchList.push_back(entity).

I designed the torch so that it can easily be used with the other game objects. Anyone on my team creating a new object, can use SetSprites() after initialising it, and torchList.push_back(), and the object will function with the torch properly.

```

bool
Entity::Initialise(Sprite* sprite)
{
    // Set all Sprites to default
    assert(sprite);
    m_pCurrentSprite = sprite;
    m_pSpriteLight = sprite;
    m_pSpriteDark = sprite;

    this->width = sprite->GetWidth();
    this->height = sprite->GetHeight();

    SetIsLit(false);
    SetIsLightable(false);

    return (true);
}

```

```

bool
Entity::SetSprites(Sprite* spriteLight, Sprite* spriteDark)
{
    // Set light and dark Sprites
    assert(spriteLight);
    m_pSpriteLight = spriteLight;

    assert(spriteDark);
    m_pSpriteDark = spriteDark;

    SetIsLit(true);
    SetIsLightable(true);

    return (true);
}

void Entity::ChangeSprites(Sprite * newSprite)
{
    // Switch current sprite
    assert(newSprite);
    m_pCurrentSprite = newSprite;
    this->width = newSprite->GetWidth();
    this->height = newSprite->GetHeight();
}

```

```

void Game::CheckTorchLight()
{
    for (size_t i = 0; i < torchList.size(); i++)
    {
        if (torchList.at(i)->GetIsLightable())
        {
            float entityX = torchList.at(i)->GetPositionX();
            float entityY = torchList.at(i)->GetPositionY();

            float torchX = torch->GetPositionX();
            float torchY = torch->GetPositionY();
            float radius = torch->GetRadius();

            if (((entityX - torchX)*(entityX - torchX) + (entityY - torchY)*(entityY - torchY)) < radius*radius)
            {
                if (torch->GetTorchOn())
                {
                    torchList.at(i)->SetIsLit(true);
                    torchList.at(i)->ChangeSprites(torchList.at(i)->GetLightSprite());
                }
                else
                {
                    torchList.at(i)->SetIsLit(false);
                    torchList.at(i)->ChangeSprites(torchList.at(i)->GetDarkSprite());
                }
            }
            else
            {
                torchList.at(i)->SetIsLit(false);
                torchList.at(i)->ChangeSprites(torchList.at(i)->GetDarkSprite());
            }
        }
    }
}

localPlayer = new Player();
localPlayer->Initialise(playerSpriteFR);
localPlayer->SetSprites(playerSpriteFR, playerSpriteDarkFR);
torchList.push_back(localPlayer);

torch = new Torch();
torch->Initialise(torchSprite);
torch->SetSprites(torchSprite, torchSpriteDark);
torchList.push_back(torch);
torchOn = false;

```

This setup worked perfectly, the only issue however is that it wasn't working for the player. I spent a lot of time trying to figure out why the player was the only object not interacting with the torch properly. Initially I thought it was due to the player class resetting its sprite somewhere else, after debugging for a long time I couldn't figure out where the issue was. I removed the player from the torchList so I could commit a working build to the SVN repository. After removing the player from the torchList the torch was no longer working. This led me to where the actual problem was, the loop for the initial CheckTorchLight was starting at 1 not 0, this was to skip the background which was part of the tileMap. Now that the torch automatically handles non-lightable objects that isn't an issue, and starting at 1 was just skipping the first object in the list. After fixing this the torch was working properly with all the entities. I tried to fix player class.

I changed the player's sprite from a tile to a free pixel art character, and made a dark version. This ended up with the player standing half through the floor. Checking the collisions I found they were hard coded

boundaries and it wouldn't change with the sprites. I committed my working build with the new torch and gave some suggestions on better handling the player collisions.

```
//actual block
for (size_t i = 0; i < tileMap.size(); i++)
{
    if (tileMap[i]->GetIsSolid() == true)
    {
        if (checkCollision(tileMap[i], localPlayer, 0, 50, 0, 50, 0, 50, 0, 50))
        {
            //collision with block
        }
    }
}

472
473
474 bool
475 Game::checkCollision(Entity* collision1, Entity* collision2, float C1Left, float C1Right, float C1Top, float C1Bottom, float C2Left, float C2Right, float C2Top,
476 {
477     if ((collision2->GetPositionX() - C2Left) - (collision1->GetPositionX() + C1Right) < 0 &&
478         (collision1->GetPositionX() - C1Left) - (collision2->GetPositionX() + C2Right) < 0 &&
479         (collision2->GetPositionY() - C2Bottom) - (collision1->GetPositionY() + C1Top) < 0 &&
480         (collision1->GetPositionY() - C1Bottom) - (collision2->GetPositionY() + C2Top) < 0)
481     {
482         return true;
483     }
484     else
485     {
486         return false;
487     }
488 }
```

21/10

Today I added FMOD to the game, I setup the initialize method and created sound effects for the torch turning on and off. I showed the team how to add sound effects for any new actions they add in.

I added a debugging mode. The player can start debugging mode by pressing ` . Once in debugging mode they will have access to cheats. Currently I have only added an infinite battery level as theres no other stats or lives added yet.

I then implemented SDL_TTF into the game. We will be using this to display the HUD. Currently only the battery level has been coded and the player health isnt added yet. I setup the functionality for TTF and added a HUD showing the battery level. I made sure to update the team on the additions and show them how they can add their own debugging features and HUD display to their parts once they are working.

Battery: 87



KKieraNN Today at 11:00 AM

Ok Just committed some progress

Added TTF and battery HUD

and debug mode

11:02 AM

```
56 {
57     if (event.key.keysym.sym == SDLK_BACKQUOTE)
58     {
59         if (game.debugMode)
60         {
61             game.debugMode = false;
62             game.infiniteBattery = false;
63         }
64         else
65         {
66             game.debugMode = true;
67         }
68     }
69     if (event.key.keysym.sym == SDLK_9)
70     {
71         if (game.debugMode)
72         {
73             if (game.infiniteBattery)
74             {
75                 game.infiniteBattery = false;
76             }
77         }
78     }
79 }
80
81 public:
82     // Control Variables
83     bool playerMoveLeft;
84     bool playerMoveRight;
85     bool m_torchOn;
86
87     // Debug Variables
88     bool debugMode;
89     bool infiniteBattery;
90 }
```

```
271 // Check Debugging Features
272 if (infiniteBattery)
273 {
274     torch->SetBatteryLevel(torch->GetMaxBattery());
275 }
276
277 setDeltaTime(deltaTime);
278 }
279 }
```

You can add debug features for any of your guys parts

Make a bool in game.h and then toggle it in the input handler

We can check for all of them at the bottom of the Process() function in game.cpp



KKieraNN Today at 7:48 PM

When the player health system is setup you can print it easily in the draw function

```
// Display Battery Level
char const* batteryHudText = "Battery:";
m_pBackBuffer->DrawText(10, 10, batteryHudText);
std::string s = std::to_string((int)torch->GetBatteryLevel());
char const* batteryLevelText = s.c_str();
m_pBackBuffer->DrawText(120, 10, batteryLevelText);

backBuffer.Present();
```

Just follow this layout

The first DrawText prints the label

The second one prints the actual BatteryLevel

You can use the to_string function to change the values to a string

After this I started working on traps. The traps are meant to be hidden in the dark and revealed by the torch. I was able to add traps using the map.h grid, initially this was getting an error, and I saw that the draw function was trying to print every tile that wasn't tileType 0. This works until trying to use the map

to place objects. I used tileType 4 in the grid for the map, so after placing a trap there is still an empty tile, instead of coding the draw function to check for solid blocks it draws everything except 0. As a work around after placing a trap I set the tileType back to 0. This will need to be done everywhere the map is used to place an object that isnt a drawable tile.

```
case 4:
    // Create Trap object on top of tile
    t = new Trap();
    t->Initialise(trap, tileMap.at(i)->GetPositionX(), tileMap.at(i)->GetPositionY() + 25);
    t->SetSprites(trap, trapDark);
    torchList.push_back(t);
    trapList.push_back(t);

    // Set tile to 0
    tileMap.at(i)->SetTileType(0);
    break;
```

After solving that issue I went to code the trap triggering on collision with the player. To do this I created an IsCollidingWith function in the entity class, and loop through the list of traps checking if they are colliding with the player. The players death hasnt been added yet so I currently just print a log message "trap triggered" and it works fine. I can add the damage to the player once the players health is finished.

```
void Game::CheckTraps()
{
    for (size_t i = 0; i < trapList.size(); i++)
    {
        if (!trapList.at(i)->IsDead())
        {
            if (localPlayer->IsCollidingWith(*trapList.at(i)))
            {
                system->playSound(soundTrap, 0, false, &channel);
                // TODO: Add player death to trap
            }
        }
    }
}
```

After getting the traps to work I was able to quickly switch the hard coded player collisions to use the IsCollidingWith function. This works a lot better and the collisions are now properly lined up with the player sprite. It doesnt handle colliding with the sides of blocks currently, it just checks if the players touching a block and will stop him falling. After this I gave some suggestions on how to handle the block collisions better.

```
//actual block
for (size_t i = 0; i < tileMap.size(); i++)
{
    if (tileMap[i]->GetIsSolid() == true)
    {
        if (localPlayer->IsCollidingWith(*tileMap.at(i)))
        {
            detectTouchGround = true;
        }
    }
}
```



KKieraNN Yesterday at 12:51 PM

Basically when he touches a block it counts as him being on the ground



maaka Yesterday at 12:51 PM

Right



KKieraNN Yesterday at 12:51 PM

When it touches the block, you can see if the player's Y > the Block's Y coordinate

If it is, the player is on top of it

If it isn't, that means he's touching the side

You should be able to copy the `isCollidingWith` function

and where it returns true

you can check if the player is on top of it

then either make him on the ground

If it's not above it, you probably need to check if he's on the left or right

by checking the X coordinates

then just move him -10 left or right or something

to bounce him off the side blocks when he walks into them

I did something similar for my individual game but it didn't have to check gravity



maaka Yesterday at 12:53 PM

Right, I will have a go on it now

Took break for lunch. I am now working on adding ladders into the game. The ladders need to be able to check if the player is touching them, and if the player is holding W it will climb the ladder. climbing

will have to cancel any current jump and jumping while climbing will cancel the climb, and require the player to press W again to grab the ladder.

I created a new movement function for the player, PlayerClimbing(). This handles the climbing movement. I use a similar function to the traps to check if the players colliding with a ladder.

```
bool Game::CheckLadders()
{
    bool onLadder = false;
    for (size_t i = 0; i < ladderList.size(); i++)
    {
        if (!ladderList.at(i)->IsDead())
        {
            if (localPlayer->IsCollidingWith(*ladderList.at(i)))
            {
                onLadder = true;
            }
        }
    }
    return (onLadder);
}
```

To allow the player to Jump I had to check how the jump mechanic works. If the player is on the ground they can jump, which is why I couldnt jump from the ladder. I added a check to make sure the player is climbing, and on a ladder. If they are they count as on the ground and can jump properly.

```
if (playerClimbing)
{
    if (CheckLadders())
    {
        detectTouchGround = true;
    }
}
```

Adding the ladders to the map was the same as adding the traps. I used tile 5 to place a ladder at that tiles location, then set the tile back to 0 so it wasnt handled by the tileMap functions.



22/10

I started today by adding more sound effects for the player jump, trap collision, and reaching the objective. When that was working I moved the FMOD Initialization to a separate function to keep the source code tidy.

```
void
Game::InitializeSounds()
{
    // Create Fmod System
    result = FMOD::System_Create(&system);

    if (result != FMOD_OK)
    {
        LogManager::GetInstance().Log("FMOD error! (%d) %s\n");
        exit(-1);
    }

    // Initialize Fmod System
    result = system->init(32, FMOD_INIT_NORMAL, 0);

    if (result != FMOD_OK)
    {
        LogManager::GetInstance().Log("FMOD error! (%d) %s\n");
        exit(-1);
    }

    system->createSound("assets\\sounds\\torchOn.wav", FMOD_DEFAULT, 0, &soundTorchOn);
    system->createSound("assets\\sounds\\torchOff.wav", FMOD_DEFAULT, 0, &soundTorchOff);
    system->createSound("assets\\sounds\\jump.wav", FMOD_DEFAULT, 0, &soundJump);
    system->createSound("assets\\sounds\\trap.wav", FMOD_DEFAULT, 0, &soundTrap);
    system->createSound("assets\\sounds\\objective.wav", FMOD_DEFAULT, 0, &soundObjective);

    playJumpSound = false;
}
```

After that I went on to add the main menu and pause menu. I started by creating button sprites, I did this by editing the tiles and putting words inside. To handle the menu functionality I created some game state variables. These keep track of the current state of the game.

```
// Initialize Game States
```

```
inMainMenu = true;
```

```
inInstructions = false;
```

```
inGame = false;
```

```
isPaused = false;
```

```
gameOver = false;
```

```
SetMenuClick(false);
```

```
torch->Process(deltaTime);
```

```
if (inMainMenu)
```

```
{
```

```
    CreateButtons();
```

```
    // Toggle Sprites When Hovered
```

```
    if (buttonPlay->IsCollidingWith(*torch)) { ... }
```

```
    else if (buttonInstructions->IsCollidingWith(*torch)) { ... }
```

```
    else if (buttonExit->IsCollidingWith(*torch)) { ... }
```

```
    // Handle Menu Clicks
```

```
    if (menuClick)
```

```
    {
```

```
        menuClick = false;
```

```
        if (buttonPlay->IsCollidingWith(*torch))
```

```
        {
```

```
            inMainMenu = false;
```

```
            inGame = true;
```

```
        }
```

```
        else if (buttonInstructions->IsCollidingWith(*torch)) { ... }
```

```
        else if (buttonExit->IsCollidingWith(*torch)) { ... }
```

```
    }
```

```
}
```

```
if (inGame)
```

```
{
```

```
    if (isPaused)
```

```
    {
```

```
        CreatePauseButtons();
```

```
        // Toggle Sprites When Hovered
```

```
        if (buttonPlay->IsCollidingWith(*torch)) { ... }
```

```
        else if (buttonExit->IsCollidingWith(*torch)) { ... }
```

```
        // Handle Menu Clicks
```

```
        if (menuClick) { ... }
```

```
    }
```

```
    else if (gameOver)
```



I setup the buttons to change to light mode based on collisions with the torch. I setup the torch movement to function in the menu but skip the draw phase (my cursor is on the play button but doesnt render in the screenshot). To handle the menu clicks, I check for left mouse button up and down, then check the current game state to determine how to handle the click. On mouse down, the torch functions are skipped if its in the menu, paused, or game over. The actual button functions are handled by mouse button up, this has been the standard way I see most buttons interact with the mouse. It allows the user to move their mouse off the button after clicking down if they change their mind or press the wrong button.

```

else if (event.type == SDL_MOUSEBUTTONDOWN)
{
    if (event.button.button == SDL_BUTTON_LEFT)
    {
        if (!game.inMainMenu && !game.isPaused && !game.gameOver)
        {
            if (event.button.button == SDL_BUTTON_LEFT)
            {
                // Turn on torch and play sound
                game.torchOn = true;
                game.PlayTorchSound();
            }
        }
    }
}

else if (event.type == SDL_MOUSEBUTTONUP)
{
    if (event.button.button == SDL_BUTTON_LEFT)
    {
        if (game.inMainMenu || game.isPaused || game.gameOver)
        {
            game.SetMenuClick(true);
        }
        else
        {
            if (event.button.button == SDL_BUTTON_LEFT)
            {
                // Turn on torch and play sound
                if (game.torchOn)
                {
                    game.torchOn = false;
                    game.PlayTorchSound();
                }
            }
        }
    }
}
}

```

To add the pause menu I had to adjust the handling of ESC press. If the game is in the main menu, pressing ESC will exit the game. When in game it will open the pause menu, if in the pause menu it will resume the game. If its game over, ESC will exit the game as well. I also had to create a ResetGame function to reset the game to its initial settings when the player exits to the main menu.

```

if (event.key.keysym.sym == SDLK_ESCAPE)
{
    if (game.inMainMenu)
    {
        game.Quit();
    }
    else if (game.inGame)
    {
        if (game.gameOver)
        {
            game.Quit();
        }
        else if (game.isPaused)
        {
            game.isPaused = false;
        }
        else
        {
            game.isPaused = true;
        }
    }
    else
    {
        game.Quit();
    }
}

void Game::ResetGame()
{
    // Initialize Debug Variables
    debugMode = false;
    infiniteBattery = false;

    InitializeSounds();

    inMainMenu = true;
    inInstructions = false;
    inGame = false;
    isPaused = false;

    gameOver = false;
    SetMenuClick(false);

    torch = new Torch();
    torch->Initialise(torchSprite);
    torch->SetSprites(torchSprite, torchSpriteDark);
    torchList.push_back(torch);
    torchOn = false;

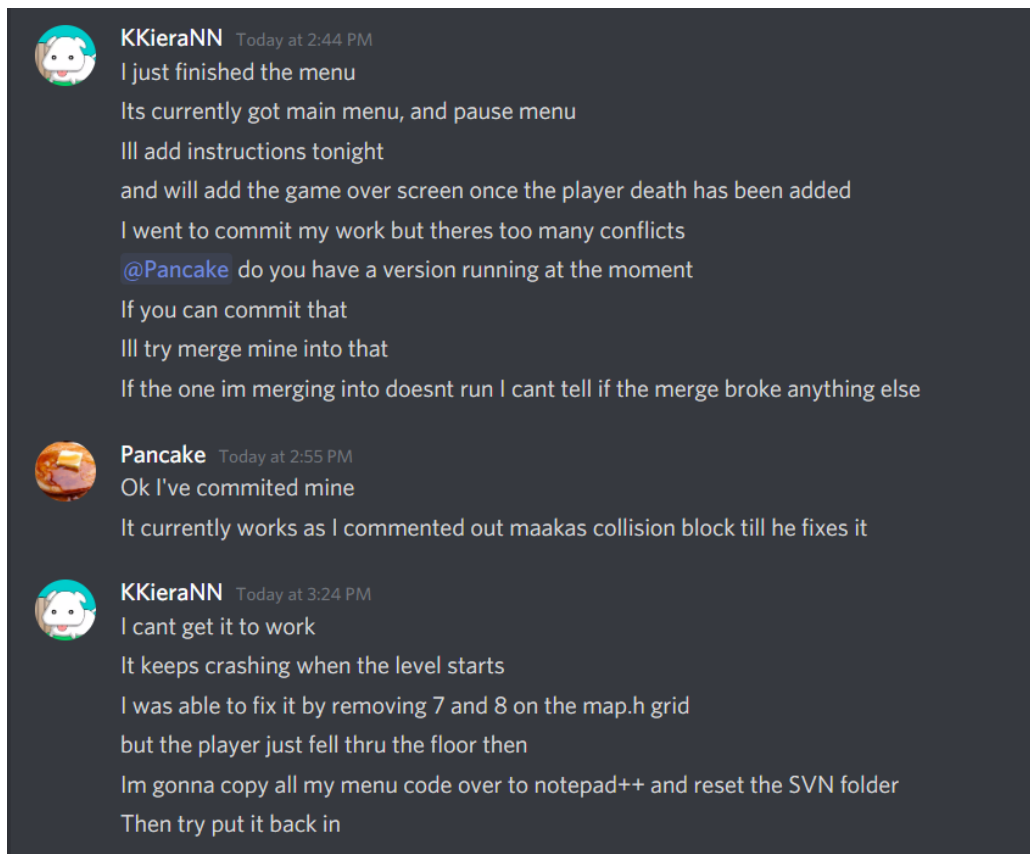
    localPlayer = new Player();
    localPlayer->Initialise(playerSprite);
    localPlayer->SetSprites(playerSprite, playerSpriteDark);
    torchList.push_back(localPlayer);

    meleeEnemyManager = new EnemyManager();
    meleeEnemyManager->Initialise(enemySprite);

    m_pMap = new Map();
    std::vector<Tile*> tileMapScroller;
    std::vector<Tile*> tileMap;
    background = new Sprite;
    background = m_pBackBuffer->CreateSprite("assets\\backgrounddark.png");
    currentPlayerStatus = onAir;
    m_pMap->SetLevel(0);
    CreateLevel();
}

```

When this was all working on my computer, I went to commit it to the repository. The player animation had broken our teams shared directory and the game wouldnt build. I tried to merge the code but there were too many conflicts, and after merging I wouldnt be able to check if it broke anywhere because the game wouldnt build anyway with the broken player animation. My solution was to copy all my menu code to notepad++, and reset my repository when another teammate had a version working (he had to comment out the broken code).



After getting a running build working, I was able to integrate my menu code back into the new version and it was working properly again.

After that I added the instructions to the main menu. Instead of creating a whole new menu for the instructions, I just created the instructions/controls as an object with no functions, and toggle the drawing of the of it when the button is pressed.



23/10

Today is the final submission day for the game. We are presenting the game in class at 4pm, the submission is due at 4:20pm. I leave home at 2:30pm to get class so I wont be able to build or work on the game after that. I have offered to build the game ready for submission if its ready before that but if anyone is still working on it I wont be able to.

The player collisions are still buggy, and the enemies dont have a detection radius so they permanently chase the player, and can trap him into a spawn killing loop. Currently there are invisible ghost enemies that you can only see with the torch and a shooting enemy (bullets are buggy). Theres no standard mele enemy that walks on the platforms as well. The map functioning well with a few levels, the next level is loaded when the current one is finished. The player has no way to win or die yet.

Last night, after committing the menu's the version in the group repository was working fine. When I tested it this morning it was extremely laggy and crashed half the time. Im not sure where the problem is and I dont have time to check, as I have to finish other assignments today.

My current additions to the game:

- Torch (left click on/off, lights entities)
- FMOD added
- Sound effects (torch, jump, trap, objective)
- TTF added
- HUD (battery level)
- Ladders (climb with W, climb cancels jump, jump cancels climb)
- Main menu
- Pause menu
- Instructions

When I left home to catch the bus into uni, my team was still working on the game so I couldnt build it for submission. The bus was late and I got to uni right as the class started, to find out my team wasnt able to build the game properly. They submitted the game without including a debug and release build, and only the solution can load the game, this also runs from the SVN shared repository so it wont be able to work on the markers computer. We will have to resubmit a working version of the game tomorrow before 4:00pm, and get -5% on our grade.

24/7

The team has been working to fix their parts of the game, I havnt made any changes except fixing an error where someone had removed the torch from the draw function. I again offered to build the release version and set up the submission folder but Maaka has said he is going to do that after they have finished fixing their issues. At 3:30pm, I updated the repository and the current build wasnt working, the teams trying to sort it out now for submission.

I have reverted the last working build I had on thursday night, before the lag issues, I couldnt get any newer ones to run and submission is in 30 minutes. I am planning to submit this version and if my team can get the new one working and submit that it will be taken as the newer version.