OneContribution Class Diagram

Entity.h

```
enum EntityType
{
        ENEMY,    KNIGHT
};

enum Direction
{
        LEFT,RIGHT,UP,DOWN,NONE
};

enum Action
{
        DYING,SPAWNING,ATTACKING,
        TAKING_DAMAGE
};

class Path
{
public:
        std::vector<sf::Vector2f>* getTiles();
        void setTiles(std::vector<sf::Vector2f>
tiles);
        void addTile(sf::Vector2f tile);
        void removeTile(int index);
        sf::Vector2f* getCurrentTile();
        int getCurrentTileNumber();
        void setCurrentTile(int index);
        bool isPaused();v
        bool isStopped();

        sf::Vector2f* getPreviousTile();
        sf::Vector2f* getNextTile();

private:
        std::vector<sf::Vector2f> m_tiles;
        int m_currentTile = -1;
};

class Entity : public sf::Drawable
{
public:
        Entity();
        Entity(EntityType entityType,
        sf::Vector2f location);
        int VecToInt(sf::Vector2i v);
        sf::Vector2i IntToVec(int i);
        ~Entity();
        int getParentDir(sf::Vector2i parent,
        sf::Vector2i child);
        void newBFS();
        void BFS();
        void tick();
        void update(sf::Time deltaTime);
        void setSpriteSheet();
        Path* getPath();
        void setPath(Path* newPath);
        void startPathing();
        void resetPathing();
        void stopPathing();
        void setHealth(int health);
        int getHealth();
        std::string getName();
        void setName(std::string name);
        bool willCollide(sf::Vector2f position);
        bool isHitting(sf::Vector2f position);
        void setSelected(bool selected);
        bool isSelected();
        void Entity::setTarget(sf::Vector2i);
        //Move an entity to a position in target
        seconds
        void moveTo(sf::Vector2f position, int
        seconds);
        bool isDead();
        bool isControllable();
        void setControllable(bool control);
        bool isVisible();
        void setVisible(bool visible);
        sf::FloatRect getGlobalBounds();
        Direction getFacing();

        sf::Text& getTextName();
        sf::Vector2f getSpritePosition();
        //sf::RectangleShape m_rectangle;

        //m_sprite stuff here
        sf::Sprite m_sprite;
        sf::Texture m_characterSprite;
        sf::RectangleShape m_
        characterSelectionBox;
        bool m_isCharacterSprite;
        void updateSprite();
        int movingFrames;
        int idleFrames;
        int deathFrames;
        int actionFrames;
        int m_curFrames;
        sf::Vector2i startMove;
        sf::Vector2i startIdle;
        sf::Vector2i startDeath;
        sf::Vector2i startAction;
        sf::Vector2i m_curStart;
        sf::Vector2i m_frameSize;
        //m_sprite stuff ends


private:


        bool m_controllable;
        EntityType m_entityType;
        int m_health;
        bool m_visible;
        std::string m_name;
        int m_maxHealth;
        HealthBar* m_hpBar;
        Direction m_facing;
        sf::Font m_font;
        sf::Text m_textName;
        bool m_isSelected;
        Path* m_path;
        //Used for getting the entities direction
        sf::Vector2f m_lastPos;
        sf::Vector2i m_target;
        void draw(sf::RenderTarget& target,
        sf::RenderStates states) const;
};
```

AnimationSprite.hpp

```
class AnimatedSprite : public sf::Drawable, public
sf::Transformable
{
public:
        explicit AnimatedSprite(sf::Time frame-
Time = sf::seconds(0.2f), bool paused = false, bool
looped = true);

        void update(sf::Time deltaTime);
        void setAnimation(const Animation&
        animation);
        void setFrameTime(sf::Time time);
        void play();
        void play(const Animation& animation);
        void pause();
        void stop();
        void setLooped(bool looped);
        void setColor(const sf::Color& color);
        const Animation* getAnimation() const;
        sf::FloatRect getLocalBounds() const;
        sf::FloatRect getGlobalBounds() const;
        bool isLooped() const;
        bool isPlaying() const;
        sf::Time getFrameTime() const;
        void setFrame(std::size_t newFrame,
        bool resetTime = true);

private:
        const Animation* m_animation;
        sf::Time m_frameTime;
        sf::Time m_currentTime;
        std::size_t m_currentFrame;
        bool m_isPaused;
        bool m_isLooped;
        const sf::Texture* m_texture;
        sf::Vertex m_vertices[4];

        virtual void draw(sf::RenderTarget&
        target, sf::RenderStates states) const;
};
```

OneContribution.cpp

```
Game* game = new Game();
Entity* ent = Game::instance()-
>getWorld().spawnEntity();
game->run();
```

World.h

```
class World : public sf::Drawable
{
public:
        World();
        void setWorld(sf::Vector2i tileSize,
        sf::Vector2i worldBounds);
        //World::World(sf::Vector2i tileSize,
        sf::Vector2f worldBounds);
        ~World();

        Entity* spawnEntity(EntityType
        type, sf::Vector2f location);
        std::vector<Entity*>& getEntities();
        sf::Vector2i getTile(sf::Vector2i
        location);

        sf::Vector2i getTilePos(sf::Vector2i
        location);
        const int getTileCount();
        sf::Vector2i getBounds();
        int getWidth();
        int getRows();
        int getHeight();
        int getColumns();

        std::list <sf::Vector2i>
        getNeighbours(sf::Vector2i);

        void tick();
        void update(sf::Time deltaTime);


        void draw(sf::RenderTarget& target,
        sf::RenderStates states) const;
        sf::Texture m_texture;
        std::vector<Entity*> m_entities;
        sf::Vector2i m_tileSize;
        sf::Vector2i m_worldBounds;

private:

};
```

Game.h

```
class Game

public:
        Game();
        ~Game();
        void run();

        UI* getUi();
        tmx::MapLoader*
getMapLoader();
        World& getWorld();
        AnimationManager*
getAnimator();

        static Game* instance();

        void setTest(std::string test);
        std::string getTest();


        //static World& getWorld();
private:
        sf::RenderWindow m_window;
        World m_world;
        sf::View m_view;
        sf::View m_miniMap;
        sf::RectangleShape m_
miniMapSprite;
        bool m_fullscreen = true;

        std::string m_test;

        static Game* m_instance;

        AnimationManager* m_
        animator;

        tmx::MapLoader* m_ml;

        sf::Music m_music;

        UI m_ui;

        sf::Clock m_tickTimer;
        //tickTimer interval in
        milliseconds
        const int m_tickRate = 20;
        //Game speed multiplier
        const sf::Time m_timePerFrame
= sf::seconds(1.f / 60.f);

        void handleEvents();
        void beginDraw();
        void endDraw();
        void tick();
        void update(sf::Time
deltaTime);

        void toggleFullscreen();
```

AnimationManager.h

```
class AnimationManager
{
public:
        AnimationManager();
        ~AnimationManager();

        sf::Texture* getTexture(EntityType type);
        void setTexture(EntityType type,
        sf::Texture newTexture);

        Animation* getAnimation(EntityType
        type, std::string animationName);
        void registerAnimation(EntityType
entityType, std::string animationName, Animation
animation);

        Animation*
generateAnimation(sf::Texture& texture, int row, int
rowHeight, int rowWidth, int frameCount);
private:
        std::map<EntityType, sf::Texture> m_
textures;
        std::map<EntityType,
std::map<std::string, Animation>> m_animations;
};
```

Component folder

HealthBar.h

```
class HealthBar : public UIComponent
{
public:
        HealthBar();
        ~HealthBar();
        int getWidth();
        void setWidth(int width);
        void setHealth(int health);
        void
        setPosition(sf::Vector2f
position);
        bool isVisible();
        void setVisible(bool
visible);

private:
        int m_width;
        int m_health;
        sf::RectangleShape m_
healthGreen;
        sf::RectangleShape m_
healthRed;
        bool m_visible;
        void
        draw(sf::RenderTarget&
        target, sf::RenderStates
        states) const;
        void
        update(sf::RenderWindow&
window);
};
```

BasicComponent.h

```
class BasicComponent : public UIComponent
{
public:
        BasicComponent();
        ~BasicComponent();

private:
        sf::Font font;
        sf::RectangleShape m_selectionBox;
        bool m_isFirstClick = true;
        sf::SoundBuffer buffer;
        sf::Sound sound;
        void draw(sf::RenderTarget& target,
        sf::RenderStates states) const;
        void update(sf::RenderWindow& win
        dow);
};
```

Animation.hpp

```
class Animation
{
public:
        Animation();
        void addFrame(sf::IntRect rect);
        void clearFrames();
        void setSpriteSheet(const sf::Texture&
texture);
        const sf::Texture* getSpriteSheet() const;
        std::size_t getSize() const;
        const sf::IntRect& getFrame(std::size_t
n) const;

private:
        std::vector<sf::IntRect> m_frames;
        const sf::Texture* m_texture;
};
```

UI.h

```
class Game;

class UIComponent : public sf::Drawable
{
public:
        UIComponent(std::string* name);
        ~UIComponent();
        virtual void update(sf::RenderWindow&
        window);
        std::string* getName();
        void setPosition(sf::Vector2f location);
        sf::Vector2f getPosition();
private:
        std::string* m_name;
        sf::Vector2f m_position;
        void draw(sf::RenderTarget& target,
        sf::RenderStates states) const;
};

class UI : public sf::Drawable
{
public:
        UI();
        ~UI();
        void handleInput(sf::Keyboard::Key
        key);
        void handleInput(sf::Mouse::Button
        button);
        sf::RectangleShape m_debugOverlay;
        bool m_debugOverlayEnabled = false;
        void toggleDebugMenu();
        void update(sf::RenderWindow
        &window);
        void updateDrawTime();
        void addComponent(UIComponent*
        component);
private:
        int m_drawTime = 0;
        int m_fps = 0;
        sf::Clock m_drawTimer;
        sf::Text m_txtDebug;
        sf::Font m_fontArial;
        std::vector<UIComponent*> m_
        components;
        virtual void draw(sf::RenderTarget&
        target, sf::RenderStates states) const;

};
```

debugGrid.h

```
public:
        debugGrid(float x, float y);
        ~debugGrid();
private:
        void draw(sf::Render
Target& target, sf::Render                States
states) const;
        void update(sf::Render
Window& window);
```