GUI Classes

Button Class

```
public:
    ButtonClass(const std::string &,int =20,int =200,int = 40);
    void setSize(int width, int height);
    void setButton(RenderWindow &);
    void setPosition(int, int);
    void setColor(const Color &);
    void setTextColor(const Color &);
    void setTextSize(const int &);
    int handleClickEvent(int, int);
    void setText(string text);
    ~ButtonClass();
private:
    RectangleShape buttonRect;
    Text buttonText;
    Font font;
```

LabelClass

```
public:
    LabelClass(string, int);
    void setPosition(int, int);
    void setLabel(RenderWindow &);
    void setTextColor(Color);
    void setText(string text);
private:
    string labelText;
    Font font;
    Text label;
```

ProgressBarClass

```
private:
    RectangleShape progressBarShape;
    RectangleShape progressBar;
    Text progressText;
    Font font;
    int progress;
public:
    ProgressBarClass(string ="", int =200, int =20);
    void setSize(int width, int height);
    void setProgressBar(RenderWindow &);
    void setPosition(int, int);
    void setColor(const Color &);
    void setTextColor(const Color &);
    void setProgress(float);
```

TextBoxClass

```
public:
    TextBoxClass(int = 20, int = 250, int = 40, int =32);
    void setSize(int width, int height);
    void setTextBox(RenderWindow &);
    void setPosition(int, int);
    void setColor(const Color &);
    void setTextColor(const Color &);
    void setTextSize(const int &);
    void setMaxStringLength(int);
    void enterTextHandler(const Event &);
    int handleClickEvent(int, int);
    std::string getContent();
private:
    std::string textString="";
    RectangleShape textbox;
    Text text;
    Font font;
    int maxTextLength;
```

Others

GameWindow

```
public:
    WorldMap *worldMap;
    RenderTools rt;
    void checkCollision(ButtonClass& interactionBtn, RenderWindow & window, int dx, int dy);
    void createGameWindow();
    void createMenuWindow();
    ~GameWindow();
    void updateProgressBar(ProgressBarClass &);
private:
    TextBoxClass *focused;
    Player *player;
    bool interactionEnabled = 0;
    Interaction interaction;
```

GameWindow includes RenderTools

RenderTools

```
public:
    RenderTools();
    Cell **cellsToRender;
    TextureBuffer textureBuffer; //buffer containing all nessecary images used while map rendering
    void drawSprite(sf::RenderWindow &, const int &, const int &, const int &,const int & xChange, const int & yChange);
    void paintWorld(sf::RenderWindow &, const int &, const int &);
    int checkHeight(const int &, const int &);
    void drawPlayerResourceGraphs(RenderWindow &window, int wood, int rock);
```

RenderTools includes TextureBuffer

TextureBuffer

```
public:
    const Texture & getTexture(const int &);
private:
    std::unordered_map<int, Texture> buffer;
    Texture loadImage(const int &);
```

Map Utils

WorldMap

```
public:
    WorldMap(int xSize, int ySize, int zSize = 0);
    int create();
    Cell ** getMatrix();
    Cell **matrix;
    Point size;
    int maxHeight;
    double progress;
    std::map<int, Player> players;
    void WorldMap::placePlayer(string, int, Player &player);
    MapObject& WorldMap::moveObject(MapObject *object, int dx, int dy);
    virtual ~WorldMap();
private:
    void WorldMap::placeMoveable(int seeds);
    void WorldMap::placeRocks(int);
    void WorldMap::placeTrees(int);
    void WorldMap::placeBedRock(int seeds, int cellsAround, int z, int ontype);
    void randomSeeds(int seeds, int z, int onType);
    int growSeeds(int count, int z, int onType);
    void fillWith(int = 0);
    void WorldMap::bringMoveablesToLife();
    int WorldMap::main(void);
    vector<Animal *> animals;
    std::thread *move;
    int isMapExists = 0;
```

Cell

```
public:
    Cell(const int & = 0,const int & = 0,const int & = 0);
    Cell(Point);
    virtual ~Cell();
    Point & getCoordinates();
    map<string, MapObject *> & getAllObjects();
    MapObject& addObject(MapObject *object);
    MapObject& getObject(string objectName);
    int contains(const string & objectName);
    int removeObject(string objectName);
private:
    Point coordinates;
    map<string, MapObject *> *objects;
```

WorldMap includes Cell

Cell includes MapObject

Point

```
public:
    Point(int = 0, int = 0, int = 0);
    ~Point();
    int x;
    int y;
    int z;
```

Player

```
public:
    Player();
    Player(int id, string playerName, int xPos, int yPos, int sprite);
    virtual int Player::interact(MapObject & player);
    virtual void Player::addItem(Item);
    map<string, Item> items;
    int makeDamage();
    int takeDamage(int dmg);
    void autoHealingThread();
    int hp = 100;
    int dmg = 10;
    virtual ~Player();
private:
    std::thread *healingThread;
```

Player Inherits from MapObject

Interaction

```
public:
    int makeInteraction(Player &player, MapObject &object, int interactionID);
    int makeInteraction(int interactionID);
    void setMapObject(MapObject &object);
    void setPlayer(Player &player);
    MapObject * getMapObject();
private:
    Player *player;
    MapObject *mapObject;
```

MapObject

```
public:
    MapObject(int objectID, string objectName, int objectSprite, Point coordinates);
    MapObject(int objectID, string objectName, int objectSprite, int x, int y, int = 0);
    virtual ~MapObject();
    virtual int interact(MapObject & player)=0;
    string objectName;
    int sprite;
    int objectID;
    Point coordinates;
    int progress();
    int elapsed = 0;
```

Animal Inherits from MapObject

Animal

```
const int ANIMAL_ID = 500;
private:
    int hp;
    int dmg;
public:
    Animal(string name, int spriteName, int x, int y, int dmg);
    virtual int Animal::interact(MapObject & player);
    map<string, Item> items;
    int makeDamage();
    int takeDamage(int dmg);
```

Item

```
public:
    string name;
    string desc;
    int amount;
    Item(string="def", string="def", int=0);
```

Rock Inherits from MapObject

Rock

```
public:
    int rockCount;
    Rock(string, int, int, int, int);
    virtual int interact(MapObject &);
private:
    const static string SOUND;
    static sf::SoundBuffer sBuffer;
    static sf::Sound sound;
    static int loadSoundBuffer;
```

Tree Inherits from MapObject

Tree

```
public:
    int woodCount;
    Tree(string, int, int, int, int);
    virtual int interact(MapObject &);
private:
    static const string SOUND;
    static sf::SoundBuffer sBuffer;
    static sf::Sound sound;
    static int loadSoundBuffer;
```