Ano Letivo: 2014/2015

Responsável: João Carlos Cardoso da Silva

TEXT ADVENTURE

Gil Fernandes 10844
Rui Caridade 09949
Jorge Paiva 10673
Pedro Barros 10669

ÍNDICE

Índice

1. Formulação do Problema 1
2. Desenho e implementação da solução 2
    2.1 Desenho 2
    2.2 Solução 3
3. Exemplos de execução 15
4. Conclusões 21
5. Bibliografia 22

1. Formulação do Problema

Quando deparados com o enunciado deste trabalho, decidimos criar um jogo de texto em que o jogador iria através de input dado pelo mesmo avançar numa história criada por nós derrotando inimigos e obter melhor equipamento.

No início a ideia era desenvolver um simples jogo que cumprisse os objetivos em cima, mas depois de algum tempo decidimos desenvolver o projeto com a intenção de evolui-lo apos a entrega do trabalho.

Para realizarmos este trabalho foi utilizado o sfml e a utilização de json para a base deste projeto. Também foi utilizada um tipo de letra do nosso agrado para utilizarmos em todas as partes do jogo e a mesma está enquadrado no estilo “old school” do jogo. Como estas ferramentas não foram abordadas no decorrer do ano letivo tivemos de realizar um extenso trabalho de pesquisa e desenvolvimento de conceitos por nós desconhecidos.

2. Desenho e implementação da solução

2.1 Desenho

Para este trabalho e usando as ferramentas do *smfl*, decidimos criar uma janela em que todas as instruções do jogo estivessem presentes em cada estado do jogo. Também foi preciso definir que opções estaria disponíveis em cada altura do jogo. O início seria numa localização previamente definida onde depois o jogador pode escolher qual a próxima localização para a qual irá mover-se.

No inicio o jogador começa a nível 1 e sem nenhum equipamento e para equilibrar os inimigos terão uma evolução simultânea com o jogador, fazendo com que quanto mais forte é o jogador mais forte e o inimigo.

As localizações, os inimigos, e os itens que o jogador pode colecionar foram armazenados em ficheiros json com as suas diversas definições que serão chamadas pelo jogo quando precisar de desenhar as localizações, os inimigos que aparecem ou os itens que são recolhidos pelo jogador aquando da derrota de um monstro ou que encontre numa localização.

Para selecionar as opções são selecionadas usando as setas do teclado e são ativados pelo Enter, sendo alterado depois ingame onde se o jogador quer dar o input de alguma ação tem de escrever qual a ação que quer desempenhar adicionando assim mais um elemento de gameplay ao jogo.

2.2 Solução

Começamos por criar o projeto com as devidas ferramentas implementadas tal como o sfml e a utilização de json. Após isto foi adicionada o tipo de letra a ser utilizado no jogo e criada uma classe genérica para listas ligadas.

A seguir foram formulados os gamestates que irão ser usados ao longo do jogo, acabando por termos 5 states. Foi também criado um statemanager (Imagem 1) que verificava a existência do state actual do jogo dando uma mensagem de erro caso fosse o contrário.

```
void StateManager::addState(const string &name, State* state) {
    if (states.find(name) != states.end()) {
        printf("Error: Specified state already exists in memory.");
        return;
    }

    states.insert(std::pair<std::string, State*>(name, state));
}

void StateManager::changeState(const string &name) {
    if (states.find(name) == states.end()) {
        printf("Error: Specified state could not be found.");
        return;
    }

    currentState = states[name];
}

void StateManager::loadState(const string &name) {
    if (states.find(name) == states.end()) {
        printf("Error: Specified state could not be found.");
        return;
    }

    states[name]->load();
}
```

Imagem 1

Nota: Algumas imagens podem não estar completas devido ao seu extenso tamanho. Para uma melhor observação consultar o código não seu devido ficheiro.

Depois foi criada a inputbox (Imagem 2) onde o jogador irá ver o jogo e fazer o input das ações a tomar, tendo precauções tal como o input do jogador ao escrever as localizações e as ações a tomar, e também conter todas as formatações tais como o tipo e tamanho de letra, as cores e as posições de cada opção.

```
InputBox::InputBox(float x, float y, sf::Font* font, int fontSize, sf::Color color) : x(x), y(y) {
    text.setFont(*font);
    text.setCharacterSize(fontSize);
    text.setColor(color);
    text.setPosition(x, y);
    string += "> ";
}

void InputBox::update(sf::Event* windowEvent) {
    if (windowEvent->type == sf::Event::TextEntered) {
        if (windowEvent->text.unicode == '\b' && string.size() >= 3) {
            string.erase(string.size() - 1, 1);
            modified = true;
        } else if (windowEvent->text.unicode < 128 && windowEvent->text.unicode != 13) {
            string += static_cast<char>(windowEvent->text.unicode);
            modified = true;
        } else if (windowEvent->text.unicode == 13) {
            sent = true;
        }
    }

    handleInput();

    text.setString(string);
    modified = false;
}
```

Imagem 2

Nesta altura foram criados os json (Imagem 3) com as armas e os tipos de armadura que o jogador pode recolher ao longo do jogo. Tendo cada ficheiro o nome dos itens em questão e a sua descrição para o jogador ler e poder saber alguma coisa sobre o item acrescentando assim mais uma mecânica de jogo que é a existência de uma história que seja adequada durante todo o jogo.

```
"Name": "Mjolnir",
"Quality": "Legendary",
"Damage": 100,
"Description": "Stay thy hand! 'Tis the God of Thunder who doth command thee!",
"Type": 3,
"BaseStrength": 90,
"BaseIntellect": 20

 "Name": "Necropolis Helm",
   "Quality": "Uncommon",
 "Description": "Minions, servants, soldiers of the cold dark, obey my call!",
   "Type": 0,
 "BaseStrength": 14,
 "BaseIntellect": 0,
 "BaseArmor": 28
```

Imagem 3

Foi também criada uma classe onde a cor dos itens difere de acordo com a sua raridade (Imagem 4), sendo os legendary com mais bónus para o jogador quando equipados que os common.

```
sf::Color Item::getQuality(const std::string &name)
{
    return quality[name];
}

void Item::loadProperties(){
    quality.emplace("Legendary", sf::Color::Color(185, 33, 33, 255));
    quality.emplace("Epic", sf::Color::Color(237, 14, 181, 255));
    quality.emplace("Rare", sf::Color::Color(205, 00, 255, 255));
    quality.emplace("Uncommon", sf::Color::Color(71, 63, 232, 255));
    quality.emplace("Common", sf::Color::Color(162, 220, 220, 255));
}
```

Imagem 4

Depois foi criada a classe Actor que gere o jogador e onde faz update dos seus atributos e gere também o state do jogador que pode entrar em stun mode (imagem 5), não podendo atacar na ronda seguinte. É também feito o update da sua vida quando recebe um ataque do inimigo.

```
void Actor::update()
{
    if (this->getAttribute(BONUS_ARMOR) == nullptr)
        MaxHp = baseHP + (this->getAttribute(STRENGTH)->getValue() * 7.50f);
    else MaxHp = baseHP + ((this->getAttribute(STRENGTH)->getValue() + this->getAttribute(BONUS_STRENGTH)->getValue()
    if (hp < 0) alive = false;
    if (resource < MaxResource){
        resource += 50;
        if (resource > MaxResource){
            resource = MaxResource;
        }
    }
    if (hp < MaxHp){
        hp += 150;
        if (hp > MaxHp){
            hp = MaxHp;
        }
    }
}
```

```
void Actor::setState()
{
    if (!stunned) stunned = true;
    else stunned = false;
}

bool Actor::takeDamage(int damage) {
    hp -= damage;

    alive = hp > 0;
    return hp <= 0;
}
```

Imagem 5

Foram criadas várias classes para gerir as habilidades, uma para gerir os diferentes tipos de habilidades e as outras para caracterizar os diferentes tipos de habilidades (Imagem 6). Sendo as habilidades que retiram vida, as que dão o estado de stun ao inimigo ou as que fazem com que o jogador receba dano inferior ao infligido.

```
int DamageAbility::getFullDamage(Actor* caster){

    int multiplierDMG = 0;
    int casterPower = caster->getAttribute(STRENGTH)->getValue() +
        caster->getAttribute(BONUS_STRENGTH)->getValue();
    int casterIntellect = caster->getAttribute(INTELLECT)->getValue() +
        caster->getAttribute(BONUS_INTELLECT)->getValue();

    switch (attributeMultiplyer)
    {
    case DamageAbility::strength:
        multiplierDMG = casterIntellect * multiplier;
        break;
    case DamageAbility::intellect:
        multiplierDMG = casterPower * multiplier;
        break;
    default:
        break;
    }

    return damage + multiplierDMG;
}

class StunAbility : public Ability
{
public:
    StunAbility(const sf::String &name, Resource resource, int cost,
        unsigned int duration, unsigned int cooldown);
    ~StunAbility();

private:
    unsigned int duration;
    unsigned int cooldown;
};
```

Imagem 6

Na classe Globals foi definido todos os resultados para o nível inicial do jogador quer para a classe de Warrior ou para a classe de Mage e também as stats do mesmo quer seja para força, intelecto ou armadura (Imagem 7). É também onde são definidos as localizações das stats no ecrã do jogo e os espaços no inventário do jogador (Imagem 8).

```
#define WARRIOR_BASE_HP 1000
#define MAGE_BASE_HP 1000

#define WARRIOR_BASE_STRENGTH 15
#define WARRIOR_BONUS_STRENGTH 0
#define WARRIOR_BASE_INTELLECT 5
#define WARRIOR_BONUS_INTELLECT 0
#define WARRIOR_BASE_ARMOR 20
#define WARRIOR_BONUS_ARMOR 0

#define MAGE_BASE_STRENGTH 5
#define MAGE_BONUS_STRENGTH 0
#define MAGE_BASE_INTELLECT 20
#define MAGE_BONUS_INTELLECT 0
#define MAGE_BASE_ARMOR 15
#define MAGE_BONUS_ARMOR 0

sf::String STRENGTH("Strength");
sf::String BONUS_STRENGTH("Bonus Strength");

sf::String INTELLECT("Intellect");
sf::String BONUS_INTELLECT("Bonus Intellect");

sf::String ARMOR("Armor");
sf::String BONUS_ARMOR("Bonus Armor");
```

Imagem 7

```
#define BATTLE_PLAYER_NAME_POSITION_X 15
#define BATTLE_PLAYER_NAME_POSITION_Y 50
#define BATTLE_ENEMY_NAME_POSITION_X 550
#define BATTLE_ENEMY_NAME_POSITION_Y 50

#define LOCATION_DESCRIPTION_POSITION_X 15
#define LOCATION_DESCRIPTION_POSITION_Y 70
#define LOCATION_NAME_POSITION_X 15
#define LOCATION_NAME_POSITION_Y 30

#define BIG_CHARACTER_SIZE 26
#define NORMAL_CHARACTER_SIZE 22

#define _BAG_MAX_SLOTS 10
```

Imagem 8

Para gerir o inventário foi criada uma classe Inventory (Imagem 9) que ficou responsável por todos os items que estão no momento em jogo quer estejam equipados no jogador ou que foram largados e foram deslocados para as bags. É também onde é desenhado o inventário do jogador e onde é gerido o bónus que os itens dão ao jogador em modo de soma de inteiros.

Alem de gerir a armadura, também gere e atualiza qual arma está selecionada e gere o inventário que o jogador pode carregar nas bags (Imagem 10).

```
void Inventory::draw(sf::RenderWindow* window, sf::Font &font){
    if (head != nullptr){
        drawText(20, 20, "Head: ", font, 24, window);
        head->draw(window, &font, 100, 20);
    }
    else drawText(20, 20, "Head: No Item", font, 24, window);

    if (chest != nullptr){
        drawText(20, 40, "Chest: ", font, 24, window);
        chest->draw(window, &font, 100, 40);
    }
    else drawText(20, 40, "Chest: No Item", font, 24, window);

    if (hands != nullptr){
        drawText(20, 60, "Hands: ", font, 24, window);
        hands->draw(window, &font, 100, 60);
    }
    else drawText(20, 60, "Hands: No Item", font, 24, window);
```

Imagem 9

```
Weapon* Inventory::getEquipedWeapon()
{
    if (weapon!=nullptr)
        return weapon;
    else{
        return nullptr;
    }
}

bool Inventory::addToBags(Item* _item){
    if (bag.getLength() + 1 > _BAG_MAX_SLOTS){
        return false;
    }
    bag.add(_item);
    return true;
```

Imagem 10

Para criar o mapa do jogo foi criada a classe World (Imagem 11)e o locationsManager (Imagem 12) onde através de um grafo com conceções entre ele, que permite ao jogador deslocar-se por entre as várias localidades e assim continuar a sua aventura. Esta classe gere tudo o que se faz em relação ao mapa do jogo, quer seja desenhá-lo, chamar as localidades do ficheiro json e adicionar ao grafo, desenhar as localidades existentes na altura que o jogador pode mover e verificar se existe caminhos com as localidades que fez load.

```
void LocationsManager::AddConnections(){
```

```
void World::draw(sf::RenderWindow* window, sf::Font& font)
{
    currentLocation->draw(window, font);
}
```

```
void World::moveTo(const std::string& name)
{
    Location* loc = getLocation(name);

    if (loc == nullptr) return;
    if (!currentLocation->hasPath(loc)) {
        std::cout << "Current location is not connected to " << name << "." << std::endl;
        return;
    }
```

Imagem 11

```
void World::addLocation(const std::string& name, const std::string& description,
    sfe::RichText displayName, sfe::RichText displayDescription, int locationLevel)
{
    displayName.setPosition(LOCATION_NAME_POSITION_X, LOCATION_NAME_POSITION_Y);
    displayDescription.setPosition(LOCATION_DESCRIPTION_POSITION_X, LOCATION_DESCRIPTION_POSITION_Y);
    Location* loc = new Location(name, description, displayName, displayDescription, locationLevel);

    locations.push_back(std::shared_ptr<Location>(loc));

    if (locations.size() == 1){
        currentLocation = locations[0].get();
    }
}

void World::connect(const std::string& start, const std::string& dest, int distance, bool twoWay)
{
    Location* startLoc = getLocation(start);
    Location* destLoc = getLocation(dest);

    if (startLoc == nullptr) {
        printf("Specified start location does not exit.");
        return;
    }
```

Imagem 12

Para as batalhas temos a classe battleManager (Imagem 13) onde é responsável por calcular o dano, aplicar ao jogador ou ao inimigo e também calcular o buff que esse dano pode acrescentar. Também é responsável por todo o visual que se vê na janela de jogo na altura de uma batalha.

```
int BattleManager::calculateDamage(int tagetArmor, int abilityDamage){
    double reduction = (tagetArmor * 0.75) / 1000 * 0.75;
    if (reduction > 0.75){
        reduction = 0.75;
    }
    return abilityDamage - abilityDamage * reduction;
}

void BattleManager::applyDamage(Actor* caster, DamageAbility* damageAbility, Actor* target)
{
```

```
void BattleStateMenu::drawText(float x, float y, const std::string&
    text, sf::Font& font, int size, sf::RenderWindow* window){
    sf::Text _text;
    _text.setPosition(x, y);
    _text.setString(text);
    _text.setFont(font);
    _text.setCharacterSize(size);

    window->draw(_text);
```

Imagem 13

Por fim podemos descrever os estados do jogo. Um dos estados é o de PostBattle em que a classe é responsável pelo que aparece na janela do jogo após cada batalha com os inimigos (Inimigos 14). Após Victória o ecrã mostra uma mensagem de sucesso e mostra ao jogador quanta experiencia ganhou. Nesta classe também é calculada a possibilidade de ser atribuído itens ao jogador, conseguindo assim melhorar a sua personagem (Imagem 15).

```
void PostBattleState::draw(sf::RenderWindow *window){

    drawText(310, 20, sfe::RichText(font) << sf::Color::Red << "VICTORIOUS!",
        CHARACTER_SIZE * 3, window);
    drawText(10, 250, sfe::RichText(font) << "You've earned " << sf::Color::
        Red << std::to_string(LAST_ENEMY_LEVEL * 100) << " xp.", CHARACTER_SIZE, window);
    drawText(25, 275 , sfe::RichText(font) << "Level: " << sf::Color::Red <<
        std::to_string(_player.LEVEL), CHARACTER_SIZE, window);
    drawText(25, 300 , sfe::RichText(font) << "XP:  " << sf::Color::Red <<
        std::to_string(_player.XP) << " (" << std::to_string(_player.TXP)<< ")"
        , CHARACTER_SIZE, window);

    if (auxloot)
    {

        drawText(10, 350, sfe::RichText(font) << "AND you've got some loot too!!!"
            , CHARACTER_SIZE, window);
        getDrop()->draw(window, &font, 10, 380);
    }
    else
        drawText(10, 350, "And nothing more...", font,CHARACTER_SIZE, window);


    drawText(15, 475, "> Continue", font, 24, window);
}
```

Imagem 14

```
Item* PostBattleState::getDrop()
{
    int auxRand = genRand();
    if (auxloot){
        for (size_t i = 0; i < GameManager::itemDatabase.size(); i++)
        {
            if (i == auxloot){
                if (GameManager::playerPtr->getInventory()->addToBags(GameManager::getItem(genRand())))
                    return GameManager::playerPtr->getInventory()->getBag().get(GameManager::playerPtr->
                    getInventory()->getBag().getLength() - 1);
                else return nullptr;
```

Imagem 15

Foi criado um estado para informar os intervenientes do projeto como tal tem o nome de créditos e mostra os nomes do grupo na janela do jogo. (Imagem 16)

```
void Credits::draw(sf::RenderWindow *window) {
```

Imagem 16

No titleMenuState (Imagem 17), é a primeira fase do jogo em que o jogador decide ver os créditos, mexer nas definições ou fazer load de um jogo anterior. E desenha tudo na janela podendo ser controlado com as setas do teclado e a partir do play, qualquer ação é desempenhada através de escrita da mesma.

```
TitleMenuState::TitleMenuState(sf::Font& font, StateManager& stateManager) : stateManager(stateManager)
{
    std::string strings[] = {
        "Play",
        "Load",
        "Options",
        "Credits"
    };

    menuOptions.resize(MENU_OPTIONS_COUNT);
    for (unsigned int i = 0; i < MENU_OPTIONS_COUNT; i++) {
        menuOptions[i] = std::make_shared<Option>(strings[i], x, y + i * menuPadding + gameTitlePadding,
            unselectedColor, menuFontSize, font);
    }

    gameTitle.setFont(font);
    gameTitle.setCharacterSize(gameTitleFontSize);
    gameTitle.setColor(sf::Color::White);
    gameTitle.setPosition(x, y);
    gameTitle.setString("Text Adventure");
}
```

Imagem 17

Durante o jogo estará a corre o GameStateMenu, que verifica se está em batalha ou não e desenha as opções que o jogador tem (Imagem 18) e a localidade onde está atualmente. Se a batalha for verdadeira muda o state para a Battle. É onde também esta situado o cálculo de random encounter (Imagem 19) onde na transição de localizações pode aparecer um monstro para derrotar. Como tal tem de fazer load dos inimigos existentes e das localizações.

```
void GameStateMenu::draw(sf::RenderWindow* window, World* world)
{
    if (auxPaths == false && auxInv == false && !auxBag)
    {
        drawText(15, 325, "> Paths", font, 24, window);
        drawText(15, 350, "> Inventory", font, 24, window);
        drawText(15, 375, "> Bags", font, 24, window);
        drawText(15, 400, "> Quit", font, 24, window);

    }
```

Imagem 18

```
bool GameStateMenu::randomEncounter()
{
    int aux;
    aux = rand() % 100;

    if (aux > 50) return true;
    else return false;
}
```

Imagem 19

3. Exemplos de execução

Na imagem 20 temos o menu inicial do jogo no qual podemos iniciar o jogo, sair, carregar o jogo e ver os créditos.

A imagem 21 corresponde ao menu dos créditos, onde pode-se consultar o nome dos criadores.

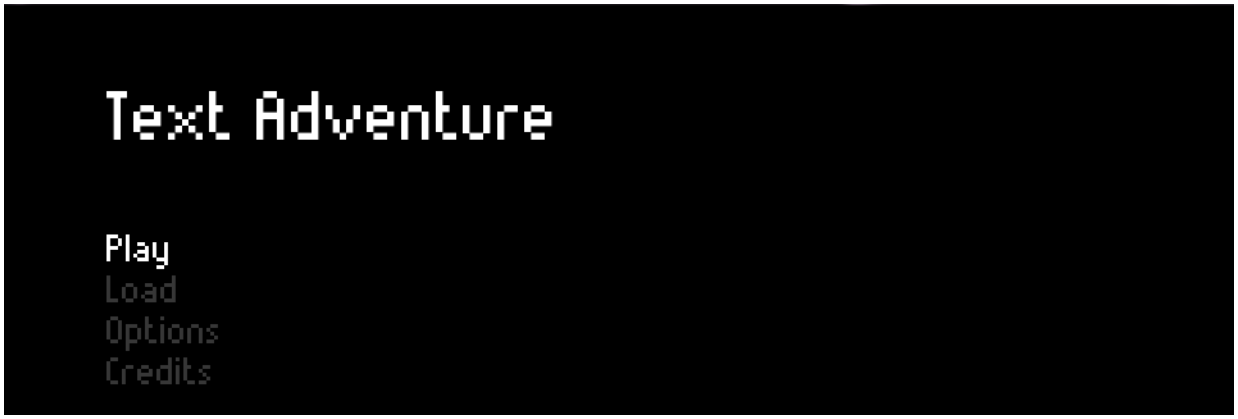

Imagem 20

Imagem 21

Na imagem 22 podemos ver a primeira localização após o início do jogo.

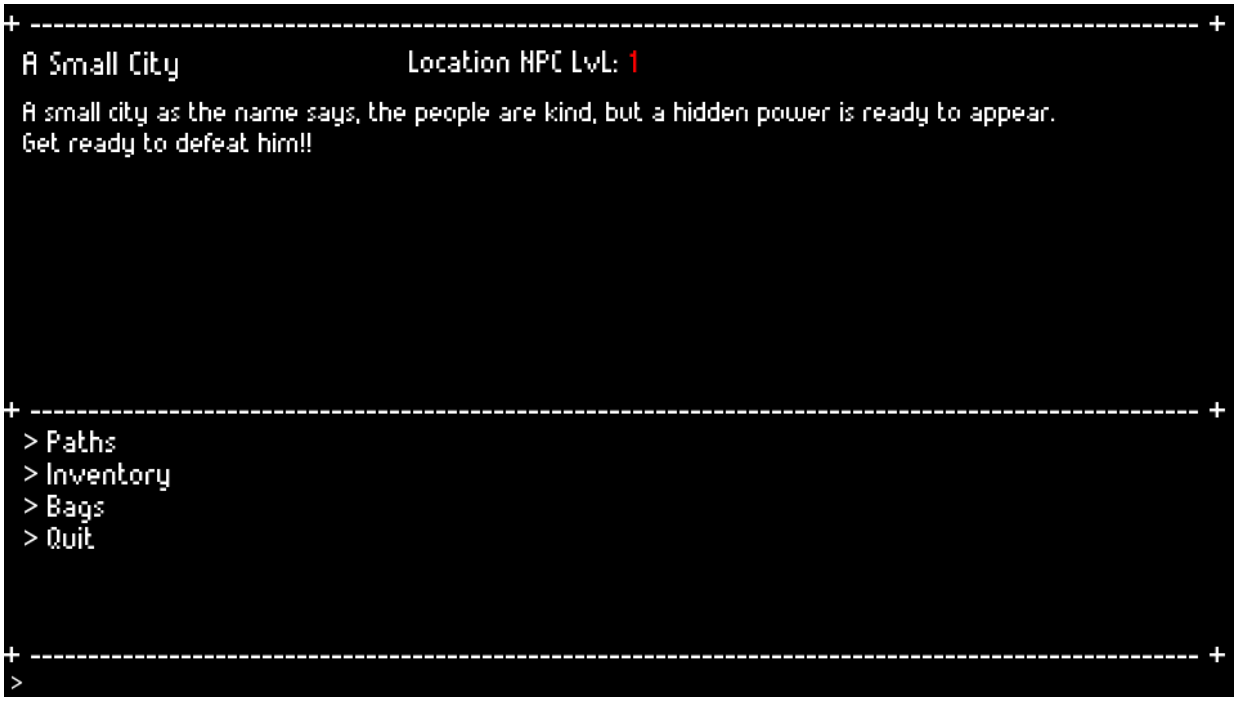

Imagem 22

Na imagem 23 podemos ver os diferentes caminhos que temos á disposição para avançarmos no jogo.

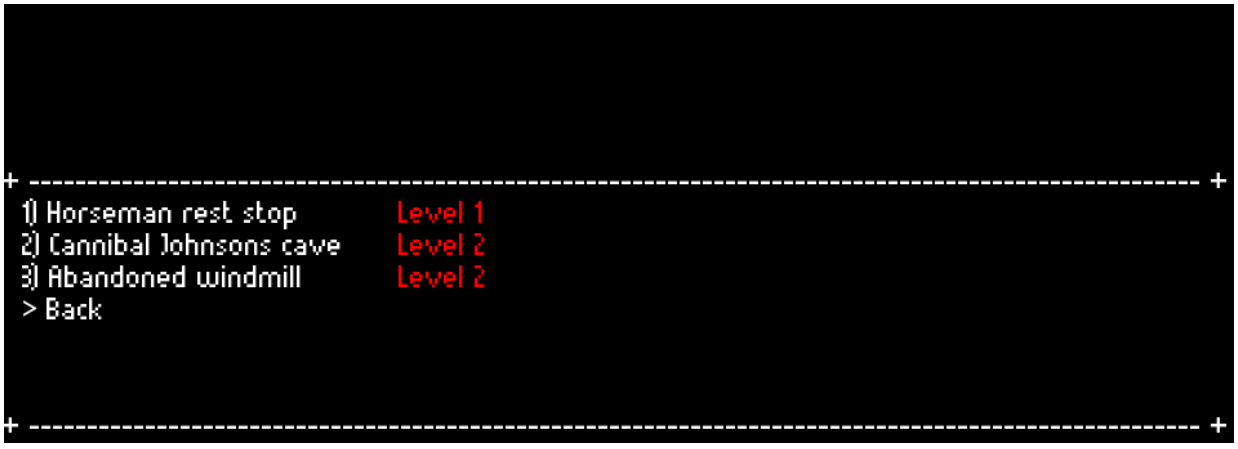

Imagem 23

Na imagem 24 podemos ver um inimigo que encontramos durante a transição de localizações, e ficamos com as opções de atacar, fugir perante o inimigo ou verificar o inventário, caso seja preciso equipar alguma coisa.

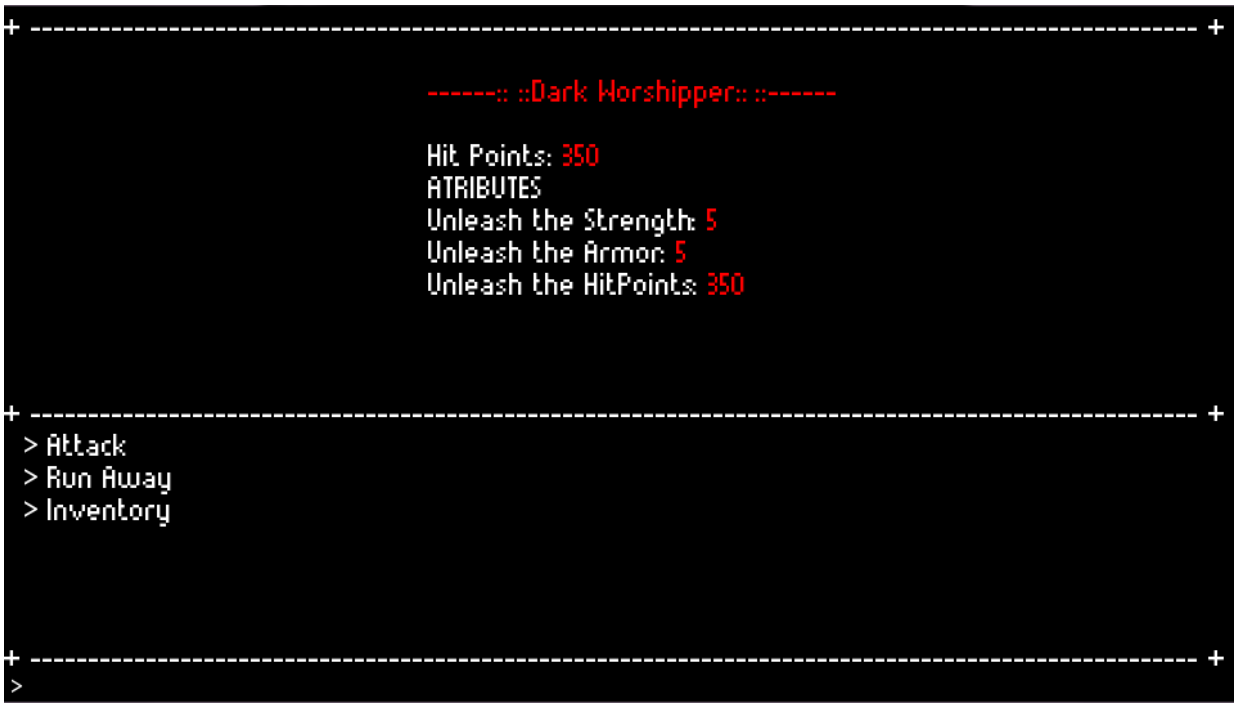

Imagem 24

Quando selecionado o modo de batalha, o ecrã muda para um estado de batalha e o jogador fica com as opções de ataque, tendo 500 de mana para utilizar no início e depois tem de gerir durante a batalha toda. (Imagem 25)

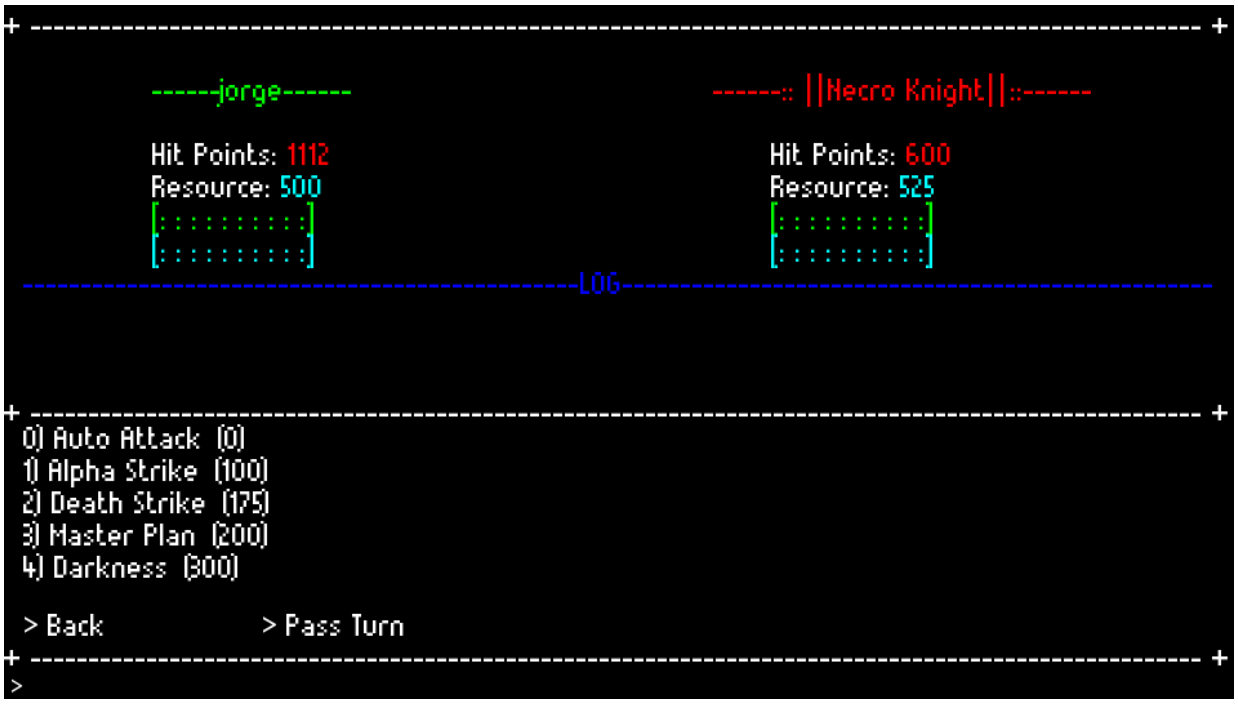

Imagem 25

Quando o jogador consegue vencer o inimigo, aparece o ecrã de vitória onde o jogador consegue ver quanta experiencia ganhou, em que nível está e quanto falta para evoluir de nível. (Imagem 26)

Neste ecrã também dá para ver se conseguiu receber algum item, como recompensa pela batalha. Podendo depois verificar o inventário e quais as stats do jogador. (Imagem 27)

Imagem 26

Imagem 27

Aqui podemos verificar uma das possíveis localizações do jogo, podendo o jogador selecionar qualquer uma das opções pois não está em nenhum estado onde as opções sejam restringidas. (Imagem 28)

Imagem 28

4. Conclusões

Ao usarmos o sfml tivemos novas possibilidades na formulação deste projeto e conseguimos criar uma interface reconhecida por muitos nós e que se assemelha muito aos jogos antigos de “text adventure”.

Ao usarmos o sfml conseguimos acrescentar novas funcionalidades ao gameplay e expandir a história e as batalhas. Este jogo está muito bem optimizado em termos de gameplay e de código, tendo tudo organizado por classes e em ficheiros de fácil alteração para os elementos chave do jogo.

Depois deste último projeto além de sabermos como formular código para um jogo também conseguimos organizar para que fique um projeto profissional e que seja facilmente entendido por alguém que não esteve envolvido no jogo.

Ao acabarmos o jogo conseguimos ter mais noções sobre outros mecanismos de desenvolvimento de jogos mas mesmo assim incorporando elementos de matéria discutidos nas aulas deste semestre. Com este projeto conseguimos juntar diversas matérias lecionadas este ano tal como listas e grafos, juntamente com o sfml para conseguirmos solucionar os nossos problemas do código e chegar a uma solução otimizada e limpa.

5. Bibliografia

1. Simple and Fast Multimedia Library

   Http://www.sfml-dev.org/tutorials/2.3/graphics-text.php

   Http://www.sfml-dev.org/tutorials/2.3/graphics-draw.php

   http://www.sfml-dev.org/tutorials/2.3/window-inputs.php

   Http://www.sfml-dev.org/tutorials/2.3/window-window.php

2. Rise Again
   Https://riseagain.wordpress.com/2012/07/19/sfml-2-tutorial-break-out/

3. Stack overflow
   Http://stackoverflow.com/questions/28461176/c-sfml-receives-close-event-just-after-window-opens

4. How to make Text Adventures
   http://www.textadventures.org/