Programação Declarativa

Trabalho 2015/2016

RPG – Dungeons & Dragons

UNIVERSIDADE DE ÉVORA

Realizado por:
João Calhau - 31621
José Pimenta - 31677

Indice

Introdução

Este trabalho enquadra-se na disciplina de Programação Declarativa, e sendo um trabalho final tem como finalidade a aplicação dos conhecimentos adquiridos ao longo desta disciplina.
Neste trabalho iremos então implementar um jogo RPG tipo “Dungeons & Dragons”, criando quartos (rooms), passagens, itens, monstros, companheiros, entre outras coisas, tendo em conta os comandos pedidos pelo professor no ponto 1. Conjunto Base, no ponto 2. Registo, no ponto 3. Modo wizard, e alguns dos comandos dos seguintes pontos 4, 5 e 6.
Os mapas serão gerados no website fornecido pelo professor (http://donjon.bin.sh/pathfinder/dungeon/) e posteriormente adaptados para remover coisas que não nos interesserão.
Este relatório irá explicar os comandos que o jogo contém, a base de dados com informações, assim como as decisões que foram tomadas ao longo da realização deste trabalho.
Iremos assim tentar dar o nosso melhor, e iremos tentar utilizar os métodos que achemos mais correctos ou propícios à boa evolução do trabalho e que no final se concretize o que nos é pedido.

0. Base de Dados, Rpg e Start

Base de dados

Na base de dados encontram-se vários objetos, entre eles os que são referenciados agora:

quarto(número, nome, andar).

passagem(quartoOrigem, quartoDestino, direção).

item(número, nome, valor). **NOTA: valor de ataque/defesa/vida**

inventario(númeroItem, nome).

itemQuarto(númeroQuarto, númeroItem, nomeItem).

needLight(númeroQuarto, númeroItem).

monsterID(letraMonstro, HP, Ataque, Defesa, Descrição).

monster(letraMonstro, VidaAtual, númeroQuarto).

companion(letraCompanion, Ataque, Descrição).

passable(númeroQuartoOrigem, númeroQuartoDestino, y/n).

player(HP, ArmaAtaqueID, ArmaDefesaID).

cPlayer(HPatual, letraCompanion).

atual(númeroQuartoAtual).

recordVar(y/n).

record(Ação).

rpg.

Comando inicial para começar o jogo.

Primeiro comando para começar tudo e que contém o ciclo de **repeat** e **fail.** Faz com que os outros comandos funcionem e faz também com que se o comando inserido seja o **stop**, que se saia para o interpretador do Prolog através de um cut **!** .

start.

Comando com informações iniciais.

Este comando é o 2º comando a ser executado, é necessário ser logo executado após o comando rpg, pois este faz com que o quarto **atual(X)** passe a ser **atual(1)** de modo a começar o jogo do inicio. Este comando também coloca o record desligado.
Para além disto tudo contém um pequeno texto inicial do jogo.
Nota: Caso use o comando **stop**, a meio dum jogo, se fizer **rpg**, irá continuar do mesmo local em que deixou, para isso não pode realizar este comando, ou irá mudar-se para o quarto 1 e desligar o record.

1. Conjunto Base

go(N).

Comando em que N é o ID dum quarto, adjacente ao quarto atual.

Primeiro verificamos qual é o quarto atual com **atual(X)**, tendo **X** como o valor do quarto atual, verificamos se há passagem do quarto **X** para o **N** (quarto que queremos alcançar), verificamos então **passagem(X,N,_)**, que caso se verifique significa que há passagem, e assim atualizamos o quarto atual para **N** , **atual(N)**, gravamos com o record se este estiver ativo, e mostramos uma mensagem ao utilizador a dizer-lhe que avançou para o quarto desejado.
Caso não exista passagem do quarto atual **X** para o **N**, o utlizador é informado com uma mensagem a dizer que não consegue ir para esse quarto e mantém-no mesmo quarto **X**.

n, s, e, w, nw, sw, ne, se, up, down.

=> direções.

1. No caso de movimentar-se no mesmo andar (floor), verifica-se o quarto em que se encontra, **atual(X)**, e sabendo **X** conseguimos verificar as passagens que partem desse quarto para outros. No caso de querer ir do quarto atual para norte, realiza-se a ação **passagem(X,Y,n)** que caso exista é porque existe uma passagem do quarto atual **X** para o quarto **Y** navegando para norte. Sendo assim, o quarto atual passa a ser **Y**, e é mostrada uma mensagem ao utilizador que a mudança foi bem sucedida e gravamos com o record se este estiver ativo. Caso não exista uma passagem de **X** para **Y** para norte, é mostrada uma mensagem ao utilizador a explicar-lhe que não existe uma saida do quarto nessa direção.(Exemplo aplica-se a todas as direções no mesmo andar, **n**,**s**,**e**,**w**,**nw**,**sw**,**ne**,**se**).

2. No caso de movimentar-se entre andares (floor), com o **up** ou o **down**, no caso do **up** verifica-se se existe passagem do quarto atual **X** para o quarto **Y** para cima, **passagem(X,Y,up)**, e se isto se verificar o quarto atual passa a ser **Y**, é mostrada uma mensagem ao utilizador e o record é aplicado se estiver ativo. Caso contrário o utlizador é informado que não há maneira de ir para cima no quarto em que se encontra.
No caso do **down** acontece o mesmo que no **up**, mas tem ainda algo mais, em que verifica se é um sitio que pode passar, **passable(X,Y,P)**, devido à existência de bosses nos quartos em que pode descer de floors, tendo de derrotar o boss desse quarto. Sendo assim down fica **passable(X,Y,y)** em que **y** é sim e **n** é não, após a derrota do boss. O resto é igual ao comando **up**.

inv.

Lista o inventário.

Comando que mostra quais os itens que o utilizador tem no inventário, para isso verificando se este tem algo no inventario para isso verificando **inventario(_K,_L)**, não interessando o que seja **_K** ou **_L**, realizando isto apenas para saber que o inventário contém algo. Sabendo se tem vamos correr todos os itens com o forall, e fazendo um print do nº do item e respectivo nome. Caso **inventario(_K,_L)** não se verifique, ou seja, não existem itens no inventário, é mostrado ao utilizador através de um print que o seu inventário não contém nada. O record é verificado se é necessário no final, para registar a ação.

drop(x).

Larga o item X (que faz parte do inventário) no quarto atual.

Comando para remover um item do inventário e colocar no quarto, tendo de para isso existir no inventário, utilizando **inventario(X,Y)** para o verificar. Caso o item exista no inventário é necessário ver qual o quarto em que se encontra para saber onde colocar o dito item do inventário, verificando-se **atual(Z)**. Sabendo que o item existe e qual o quarto em que se encontra o utilizador, adiciona-se o item ao quarto com **asserta(itemQuarto(Z,X,Y))**, sendo **Z** o número do quarto, **X** o número do item e **Y** o seu nome. É mostrada uma mensagem ao utilizador através de um print.
Caso não exista o item no inventário quando se verifica **inventario(X,Y)**, é mostrada uma mensagem ao utilizador a explicar que não pode fazer drop desse item, pois não o tem. O record é verificado se é necessário no final, para registar a ação.

get(X). e get(Z).

Retira o item X (que deverá estar no quarto atual) e coloca-o no inventário.

1. No caso do **get(X)**, irá verificar-se **X** para o número do quarto atual **Y**, vendo **atual(Y)**, verificando **itemQuarto(Y,X,Z)**, sendo **Y** o número do quarto, **X** o número do item e **Z** o nome do item. Caso se verifique, é porque o item existe no quarto **Y**, e sendo assim adiciona-se o item ao inventário no formato **inventario(X,Z)** e é mostrado ao utilizador com um print que o item foi pego. Caso o item não se encontre no quarto, é mostrada uma mensagem ao utilizador dizendo que o utilizador não consegue pegar isso, pois não existe nesse quarto.

2. No caso do **get(Z)**, acontece o mesmo que no ponto 1, apenas diferenciando que o utilizador fornece o nome do item, mantendo-se o resto **itemQuarto(Y,X,Z)**, e seguintes passos.

look.

Apresenta a descrição do quarto atual (o floor em que se encontra o nome do quarto) e diz para onde há saídas (as direções das saídas e a que quartos os leva). Indica que itens se encontram no quarto.

Quando se realiza um **look** para verificar informação acerca do quarto atual, é necessário saber qual esse quarto, vendo-se assim **atual(X)**, sendo **X** o nº do quarto atual, sabendo **X** sabe-se mais informação do quarto com **quarto(X,N,A)**, em que **N** é o nome do quarto e **A** o andar (floor).
Agora é necessário saber se existe visibilidade para ver informação do quarto (ponto extra das extensões avulsas Luz/Visibilidade) e para isso verifica-se se o quarto em que se encontra é um dos quartos em que precisa de itens especificos para saber a informação, vendo assim **needLight(X,M)**, em que **X** é o quarto atual verificado em **atual(X)** e **M** é o número do item necessário nesse andar para visibilidade (tendo cada 1 dos 3 andares um item especifico para esse andar).
Caso não seja necessário o item, é mostrada toda a informação acerca do quarto como andar, nome, todas as passgens através de **passagem(X,H,Y)** em que **H** é o número do quarto adjacente e **Y** a direção tomada, informação utilizada nos prints. São ainda mostrados os itens do quarto caso estes existam e no final uma mensagem a verificar caso exista monstro no quarto com a verificação **monster(ID, _, X)** em que **ID** é a letra correspondente ao monstro (ex: a).
Caso seja necessário o item de luz, verifica-se se no inventário se encontra esse item, usando para isso **inventario(M, _N1)**, em que **M**, verificado em **needLight(X,M)** é o número do item necessário. Se existe esse item no inventário é então mostrada a informação já referida, caso não contenha o item de luz necessário, o utilizador é informado que necessita de encontrar o item de luz no floor em que se encontra.
Como sempre o comando record se ativo é realizado.

2. Registo

record.

Indica que o interpretador deve começar a gravar os movimentos e outras ações (p/ex get e drop).

Comando que quando executado mostra a mensagem ao utilizador a dizer que começou a gravar todos os movimentos, removendo os itens gravados até ao momento, isto é, quando executado o comando record, apenas os movimentos seguintes serão gravados, e guardados, fazendo com que os anteriores não interessem. Assim é tornada a variável **recordVar(y)**, em que **y** significa que os movimentos serão gravados, isto porque quando o comando record é verificado em todos os outros comandos, vai verificar como se encontra **recordVar(R)**, em que se **R** tiver o valor **y** grava esse comando, se estiver **n**, não irá gravar, e é assim que os comandos executados são gravados.

track.

Pede ao interpretador para listar todos os passos dados desde o record.

Comando que quando executado mostra a mensagem ao utilizador a dizer todos os passos dados desde o último record, e percorre todos os **record(X)**, em que **X** é uma ação do tipo **get(K)**, **drop(F)**, ou outros.

forget.

Indica ao interpretador que deve parar de gravar os movimentos.

Comando que quando executado mostra a mensagem ao utilizador a dizer que não irá gravar mais movimentos, para isso altera a variável **recordVar(R)** para **recordVar(n)**, em que **n** significa que as próximas ações realizadas quando chegarem ao momento de fazer record não o irão fazer devido a essa variável ser **n**.

stop.

Sai do interpretador (o local fica guardado), regressando ao top level do Prolog.

Comando inserido dentro do comando **rpg.** em que caso se verifique, faz cut **!** e sai do interpretador do programa.

3. Modo “Wizard”

jump(N).

Salta para o nó com identificador N, qualquer que ele seja.

Comando que muda o nó atual para o nó N. Para isso remove-se o nó atual com um **retract(atual(_))**, e fazendo **asserta(atual(N))** . O utilizador é informado que foi teleportado para um novo quarto, e conforme o record esteja ativo ou não, a ação é guardada.

warp(X).

Faz aparecer um item X no local atual.

Verificando-se o quarto atual com **atual(Y)**, vê-se qual o item que o utilizador quer adicionar através de **item(X,K,_W)**, e depois adicionando-o ao quarto em que se encontra com **asserta(itemQuarto(Y,X,K))**.
É feito print duma mensagem informativa ao utilizador dizendo que o item foi colocado no quarto e caso o record esteja ativo, a ação é gravada.

destroy(X).

Faz desaparecer o item X que deverá estar no local atual.

Verificando-se o quarto atual com **atual(Y)**, vê-se qual o item que o utilizador quer remover através de **item(X,K,_Z)**, e depois vê se esse item se encontra no quarto através de **itemQuarto(Y,X,K)**, que caso seja verdade, seja removido com **retract(itemQuarto(Y,X,K))**.
É feito print duma mensagem informativa ao utilizador dizendo que o item foi removido do quarto caso este estivesse presente no quarto, no caso de não estar presente, o utilizador é informado que o quarto não contém o item que se pretende destruir. O record é verificado no final, como sempre.

4. Outros Comandos

Foi dada alguma liberdade no trabalho no sentido de implementarmos comandos e funções que bem vissemos interessantes e que ajudassem no desenvolvimento de um bom trabalho. No nosso caso como grupo, decidimos fazer os seguintes comandos, que serão explicados em detalhe:

equip(X).

Equipa uma arma de ataque ou uma arma de defesa no jogador.

Comando que primeiro verifica se o utilizador contém o item no inventário através de **inventario(X,Y)**, em que **X** é o número do item e **Y** o nome do item. Se o tiver no inventário, verifica qual o valor de **X**, sendo que de 6 a 10 é um item de ataque, de 11 a 15 é um item de defesa. Qualquer outro valor de **X** será um item que nem é de ataque nem de defesa e por esse motivo não poderá ser equipado e é assim enviada uma mensagem ao utilizador.
Portanto se **X** for um valor válido para ataque, o item de ataque que esteja atualmente em uso pelo utilizador, verificado através de **player(HP, Sword, Shield)**, em que **HP** é o valor de vida do jogador, **Sword** é um número dum item de ataque equipado, **Shield** é um número dum item de defesa equipado, será substituido, e assim adiciona-se o item de ataque **item(Sword,B, _C)** ao inventário através de **asserta(inventario(Sword,B))**. O item **X** é adicionado depois no player através de **asserta(player(HP,X,Shield)**, alterando o valor de **Sword** para **X**. Uma mensagem é agora mostrada ao utilizador a referir as alterações realizadas.
No caso dum item de defesa o principio é o mesmo, mas neste caso é adicionado ao inventário na forma de **asserta(inventario(Shield,K))**, e adicionado ao player o item **X** na forma **asserta(player(HP, Sword, X))**.
Ainda no caso do utilizador querer equipar um item que este não contenha, é mostrada uma mensagem a dizer que o utilizador não tem esse item que possa equipar.
O record funciona aqui como nos comandos já referidos anteriormente.
Nota: existe apenas uma instância de **player(HP,Sword, Shield)** , que representa o jogador no momento atual.

listItem.

Mostra uma lista de todos os itens no jogo.

Comando que verifica todos os itens que existem na base de dados e mostra uma mensagem com todos os itens com o seu número e nome.
O record funciona aqui como nos comandos já referidos anteriormente.

attack.

Ataca um monstro que se encontra no quarto.

Comando que verifica o quarto em que se encontra com **atual(X)**, as armas que o utilizador tem equipadas com **player(_,W,S)**, em que **W** é um item de ataque **item(W, _ , PA)**, e **S** é um item de defesa **item(S, _ , PD)**.
Seguidamente vê-se o companheiro que o jogador tem consigo no momento com **cPlayer(HP, Pet)**, em que **HP** é a vida do jogador, e **Pet** é a letra do companheiro NPC, **companion(Pet, PetAttack, PetName, _)**.
Neste momento, após verificar estes dados, verifica-se se existe um monstro no presente quarto através de **monster(ID, CHP, X),** em que **ID** é a letra que representa o monstro, **CHP** é a vida atual do monstro. Caso não exista monstro no quarto é mostrado ao jogador que não existe monstro no quarto e o comando termina, mas caso exista um monstro são executados uns cálculos para ver que dano sofre o monstro e o jogador.
Vê-se então **monsterID(ID, _, A, D, _)**, em que **ID** é a letra referente ao monstro como já foi referido, **A** é o poder de ataque do monstro e **D** é o poder de defesa do monstro. Os cálculos do ataque são calculados com a fórmula:

Pattack = PA – D
TotalAttack = Pattack + PetAttack,
NHP = CHP – TotalAttack

MAttack = A-PD
PNHP = HP-MAttack

Com os seguintes significados:

PA -> valor de ataque do item do jogador.
PD -> valor de defesa do item do jogador
A -> valor de ataque do monstro
D -> valor de defesa do monstro
Pattack -> diferença entre ataque do jogador e da defesa do monstro
PetAttack -> valor de ataque do companheiro NPC.
TotalAttack -> Soma do ataque do companheiro e da diferença entre ataque do jogador e defesa do monstro.
CHP -> vida corrente do monstro
NHP -> novo valor de vida do monstro, subtraindo da vida corrente CHP o valor sofrido no ataque TotalAttack.
Mattack -> diferença entre ataque do monstro e da defesa do jogador.
HP -> vida corrente do jogador.
PNHP -> novo valor de vida do jogador, subtraindo da vida corrente HP o valor sofrido no ataque Mattack.

Após todos estes cálculos, o utilizador é informado de quanto ataque sofreu do monstro e quando ataque infligiu no monstro.
Depois é verificado os valores de **PNHP**, que se for 0 ou menos, significa que o jogador morreu, e assim é enviada uma mensagem ao utilizador a informar que jogador morreu e perdeu o jogo – GAME OVER. O valor de **NHP** caso esteja a 0 ou menos significa que o monstro morreu, e assim é enviada uma mensagem ao utilizador a dizer que o monstro foi morto. Caso o monstro morto tenha o **ID== c**, significa que o jogador matou o Boss final, e assim é mostrada a mensagem final de como ganhou o jogo, se não for o boss final, mas for um Boss do 1º andar ou do 2º andar, é agora possivel descer na dungeon, através da alteração de **passable(X,Y,n)** para **passable(X,Y,y).**
Caso o jogador atinga o monstro e não o mate com 1 ataque que faça nem morra o jogador, ou seja, a vida do monstro não ficou menor ou igual a 0 nem a do jogador, a vida do monstro e a vida do utilizador são atualizadas após 1 ataque.
O record é aplicável como nos outros comandos.

inspectM.

Inspecciona o monstro e mostra informação sobre este.

Comando que informa o utilizador com informação do monstro, para isso verificando com **atual(X)** o quarto em que se encontra e verifica com **monster(ID,HP, X)** se existe algum monstro no quarto atual. Caso exista, verifica-se o monstro com **monster(ID, _ , A, D, Desc)** em que **Desc** é o nome do monstro, **A** é o ataque do monstro, **D** é a defesa do monstro, e **HP** é a vida do monstro.
Caso não exista monstro presente no quarto, é feito print duma mensagem a dizer que não existe nenhum monstro no quarto.
O record é aplicável como nos outros comandos.

inspectP.

Inspecciona o player e mostra informação sobre este.

Comando que informa o utilizador com informação do player, vendo assim **player(_, W, S)**, em que **W** é o número da arma de ataque, e **S** é o número da arma de defesa, **cPlayer(HP, Pet)**, em que **HP** é a vida do player, **Pet** é a letra do companheiro NPC, **companion(Pet, PetAttack, PetName, PetDescription)**, em que **PetAttack** é o ataque do companheiro, **PetName** é o nome do companheiro, e **PetDescription** é a descrição do companheiro. Vê-se ainda quais são os itens que o player tem equipados, para isso: **item(W, N, A)**, e **item(S, N1, D)**, em que **N** e **N1** é o nome do item, **A** e **D** são o valor da arma, em que **A** é usado para ataque por ser arma de ataque, e **D** é usado para defesa, por ser arma de defesa.
Com esta informação toda, sabemos tudo de importante acerca do player, sobre ele, sobre o companheiro e sobre as suas armas, e assim fazemos print desta informação toda de modo a informar o melhor possivel o utilizador.
O record é aplicável como nos outros comandos.

eat(X).

Jogador come item X de modo a aumentar o seu HP.

Comando que utilizador usa para aumentar o HP do player, para isso verifica qual o item com **item(X,Y,Z)**, e se este existe no inventário **inventario(X,Y)**, e se não existir é enviada mensagem ao utilizador a dizer que não tem esse item para o comer.
Caso tenha esse item e o valor de **X** esteja entre 1 e 5, quer dizer que é um item que se pode comer, mas caso não se verifique é enviada a mensagem ao utilizador a informá-lo que não pode comer esse item.
Existindo o item no inventário e sendo um item comestivel, o item é removido do inventário e calcula-se através do **item(X,Y,Z)**, quanto pode restaurar vida do utilizador, pois **Z** representa o valor de vida nos itens que têm **X** entre 1 e 5.
Verifica-se o valor de vida atual do jogador **HP**, obtido de **cPlayer(HP,Pet)**, e calcula-se o novo valor com **NHP = HP + Z**. Caso o valor seja maior que 10, valor máximo de vida quando curando, o valor do player fica-se por 10, noutro caso em que não passe de 10, o valor fica o novo valor calculado, ficando assim **cPlayer(NHP, Pet).**
O record é aplicável como nos outros comandos.

summon(X).

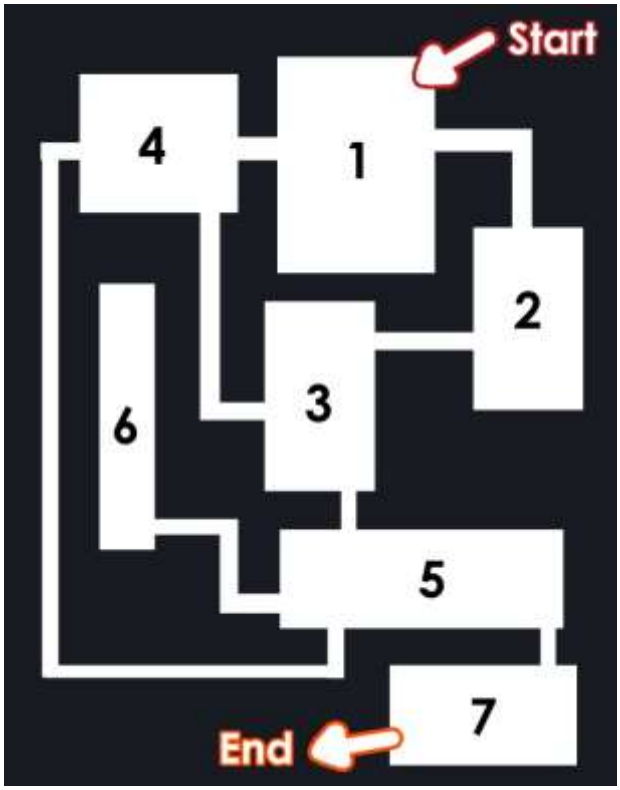
Altera o companheiro NPC atual para X.

Comando que altera o companheiro, para isso verifica se tem o item necessário para o invocar, vendo no inventário **inventario(X,Y)**, e o item correspondente **item(X,Y, _Z)** e como está o player tem atualmente com **cPlayer(HP,_Pet)** para poder ser alterado.
Caso o utilizador não tenha o item no inventário que está a tentar usar, é mostrada uma mensagem a dizer que o utilizador não possui o item que está a tentar usar para a invocação. No caso de ter o item, este tem de ter valor de **X** igual a 16,17 ou 18, valores dos itens de invocação no jogo. Conforme o valor do item assim é o companheiro que se junta ao jogador. Se for um dos valores possiveis é alterado o **cPlayer**, caso seja um valor <16 ou >18, é mostrada uma mensagem a dizer que não se pode fazer summon com esses itens.
O record é aplicável como nos outros comandos.

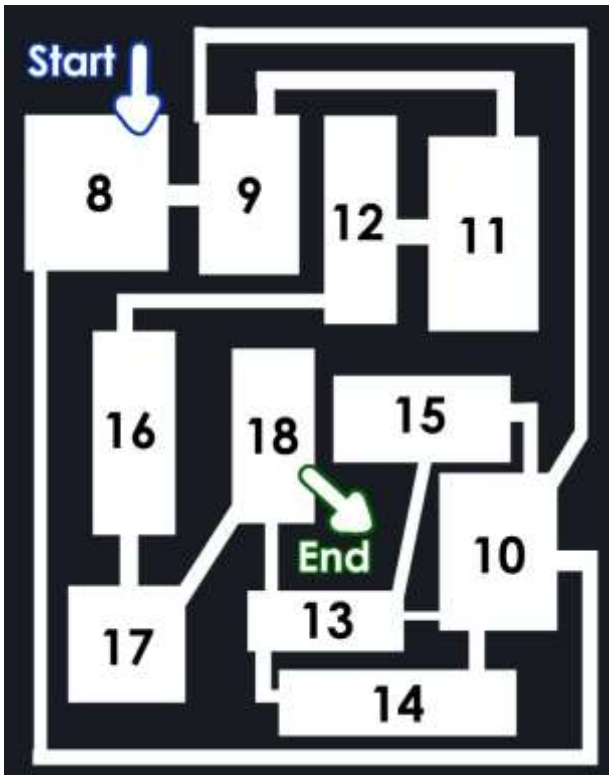
5. Anexos

Neste ponto estão algumas imagens ilustrativas de como serão os 3 floors que o jogo contém de modo a ajudar a se orientar o utilizador do jogo.

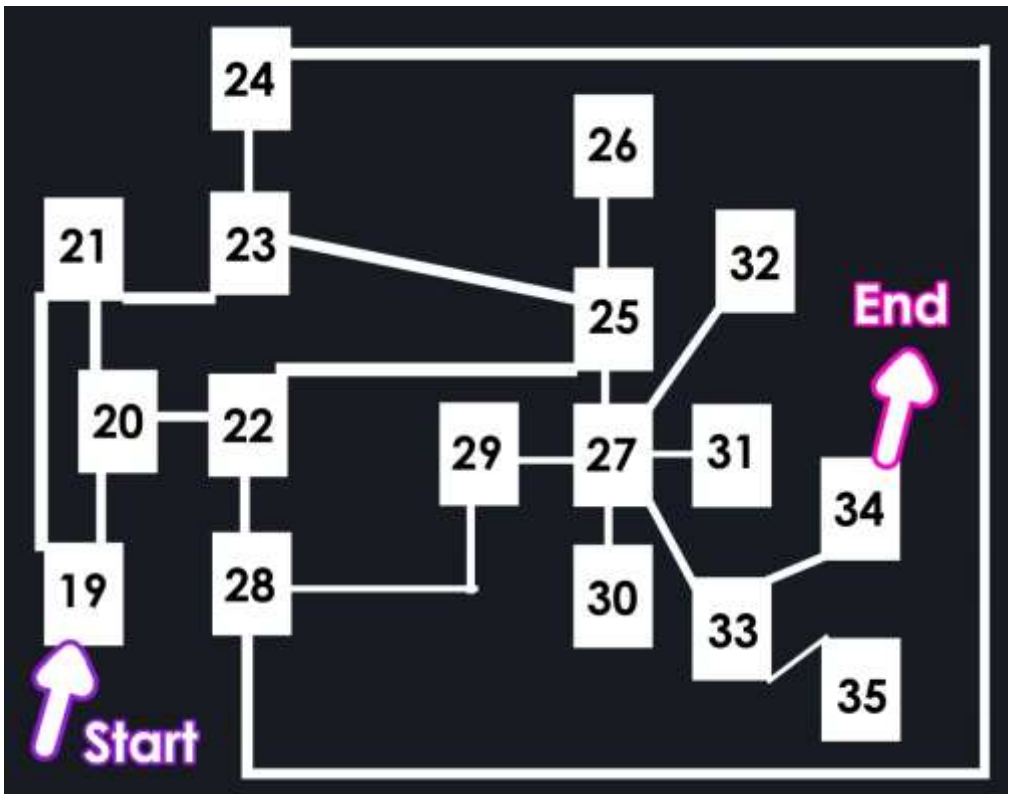
Andar 0 – Andar Superior



Andar 1 – Andar Inferior Intermédio



Andar 2 – Andar Inferior Final



6. Conclusão

Após a realização deste trabalho conseguimos aprender algumas coisas que não sabiamos ou não tinhamos tanta experiência. No geral foi um bom trabalho e divertido de fazer, em que fizemos os pontos 1,2,3 todos como pedido e fizemos alguns extras tais como NPCs, dos quais monstros que atacamos, tendo assim um sistema de combate, e NPCs que são companheiros e nos ajudam também nas batalhas, a extensão de Luz/visibilidade, na qual para obter a informação do quarto é necessário ter itens de luz especificos para o andar, a inclusão de 3 andares, sendo que começa-se no andar 0 e o andar 1 e 2 são acessíveis após uma luta contra um Boss final do andar (tendo outros monstros espalhados pelos quartos). O player pode ainda comer para restaurar HP e aumentá-lo, listar todos os itens que o jogo contém, ou utilizar os comandos inspectP ou inspectM para obter máximo informação acerca de si (jogador) ou do monstro, respectivamente.
Os mapas nos anexos mostram como os 3 andares foram projectados e dão uma ideia boa de orientação, e são assim um boa ajuda para se utilizar enquanto se joga o jogo.
Resumindo, fizemos alguns comandos que o professor sugeriu e outros que pensámos ser bons para este tipo de jogo, e no geral ficou um jogo agradável de se jogar, e com muitas possibilidades de acrescentar mais features, caso tivessemos mais tempo para as criar, mas pensamos ter um jogo sólido no estado em que este ficou.