Documentation Générale

Le projet Red fait ci contre, est un jeu dans lequel vous incarnez

un personnage et vous affrontez des monstres, en choisissant votre

classse, différents skills seront à votre dispositions, ces skills

ont un coût et une puissance phénoménale, le monde dans lequel vous

êtes s'appelle Ora, on y trouve de nombreux monstre, certains plus

forts que d'autres mais vous devrez les vaincre tous...

Histoire

Il était une fois un jeune chevalier combattant des dragons a ses

heures perdues, fou amoureux de sa femme Dana, un jour un monstre

inconnu du monde Ora apparu et devora Dana, ce monstre géant et

d'une force spectaculaire fut la raison de vivre de ce chevalier,

il était a sa traque pendant si longtemps qu'un jour il se décida

à l'éliminer, il engloutit tellement de potions que ses yeux

devinrent noir, il brandit son épée et trancha la tête du monstre

pendant un combat sanglant, désormais vous êtes ce chevalier et

votre but sera de tuer tout les monstres restant dans ce monde ...

Documentation Technique

Ce jeu fonction sous forme d'appel de fonction, en effet c'est assez

simple, dans un Menu() principal toutes les fonctions sont appelées

construites dans des fichiers.

et se sont toutes des sous menu, chacunes des fonctions sont

différents. Dans le menu, dés que le programme est lancé cela nous

affiche directement dans le terminal des choix a faire, en effet

ces choix appellent grâce a des switch case, les fonctions utiles

à la demande, ce fonctionnement permet l'appel de fonction libre.

La fonction Menu\_skill\_choix(), permet a l'utilisateur de choisir

des skills grâce a une demande fmt.Print dans laquelle une variable

int est scanné grâce a fmt.Scan et nous permet de récupérer ce choix

entré dans le terminal, selon ce choix grâce à un switch case, selon

le chiffre choisi, un sous menu se lance, de cette même manière

l'utilisateur à un choix a faire, et selon ce choix il peut accéder

à un skill, pour une certaine logique, les skills sont accessible

selon la classe de l'utilisateur, grâce a des conditions if avec

comme condition utilisant la fonction verif\_classe qui permet de

vérifier si la classe est la bonne et dans ce cas l'utilisateur peut

choisir son skill qui lui sera stocké dans la structure skill et

aura un coût en mana qui est l'argent du jeu.

La fonction Marchand(), permet d'avoir un système d'argent dans le

jeu, de la même manière que Menu\_skill\_choix(), grâce à un fmt.Print

suivi d'un fmt.Scan, selon le choix de l'utilisateur, grâce a un

switch case, dans chacune des cases, grâce à la fonction

AddInventory() qui ajoute dans l'attribut inventaire de la structure

Personnage, l'item acheté au marchand, par la suite un retrait

d'argent ce fait grâce à la fonction retrait\_monnaie() et à la fin

de la case, un retour au menu grâce à un appel : Menu().

La fonction Affichage() permet d'afficher qui en prenant en paramètre

un élément string appelé str, qui sera grâce à un fmt.Print() affiché

avec une phrase disant ('Vous avez équipé', str), cette fonction est

donc utile à appelée pour donnez l'information à l'utilisateur

qu'il est bien équipé de cet item (str).

La fonction Menu(), permet l'appel direct de toutes les fonctions

qui sont toutes des sous menus, en effet dans la fonction Menu()

la logique est la même que sur le menu\_skill\_choix(), l'exemple de

la fonction menu\_skill\_choix() est parfait puisque pour Menu(),

Menu\_skill\_choix() est un sous menu qui comme d'autres fonctions

Menu\_Forgeron(), Marchand() qui sont des sous menus a la fonction

principale Menu().

L'utilité d'un fmt.Print() est de permettre à l'utilisateur de choisir

ce à quoi il veut accéder, grâce à un fmt.Scan().package main

Suivant ce schéma technique, chaque sous menu ont eux mêmes leur sous

menus qui fonctionnent de la même manière.

Ainsi le Menu() permet à l'utilisateur de naviguer dans le jeu de la

manière qu'il souhaite.

La fonction Init() de la structure Personnage, permet à l'utilisateur

d'initialiser les informations principal du personnage, grâce à un

fmt.Print() de choisir le nom, qui ne doit pas comprendre autre chose

que des lettres en minuscule et seulement la première lettre en

majuscule.

Ensuite pour choisir la classe, nous avons créer une boulce for qui

permet à l'utilisateur de choisir qu'une seule classe puisque la

boucle à une condition d'arrêt i < 1, alors qu'on initialise i := 0,

et qu'on l'incrémente de 1 à chaque tour dans la boucle, en effet

on passe ainsi qu'une seule fois dans la boucle.

Le choix de la classe quand à lui est choisi de la même manière que

les autres fonctions utilisent un fmt.Print(), ensuite un fmt.Scan()

qui récupère le choix de l'utilisateur et grâce au switch case

print selon le case, la classe attribuée.

La fonction TakePot(), permet au joueur de prendre une potion que

l'on initialise a 20 au début de la fontion, ensuite grâce

à une boucle : for i := range p.inventaire, suivi d'une condition

if qui permet la vérification que dans l'inventaire on a bien un i

qui est égal à 'potion de vie', si la condition est respectée, on a

encore une condition qui elle permet que l'ajout de cette potion

qui vaut 20, après l'ajout au point\_de\_vie\_actuel ne dépasse pas

la limite point\_de\_vie\_maximal, ainsi la condition :

if p.point\_de\_vie\_actuel+potion > p.point\_de\_vie\_maximum

Si cette condition est respectée alors point\_de\_vie\_actuel = point\_de

vie\_maximal, en mettant les points de vie actuels égaux aux points de

vie maximal du personnage, cela évite que le personnage se retrouve

avec plus de points de vie de la limite autorisée.

Pour finir grâce à la fonction RemoveInventory(), en l'appelant on

enlève la potion de l'inventaire qui vient d'être utilisée et on

fmt.Print() les points de vie actuels du personnage.

La fonction PoisonPot(), permet au joueur grâce à une potion de poison

de diminuer les points de vie du monstre, dans la même logique que la

fonction TakePot(), on fait une boucle : for i := range p.inventaire

pour être dans l'inventaire et une condition :

if p.inventaire[i] == 'Potion de poison'

Grâce à cette condition, si il y a bien un item 'Potion de poison'

les points de vie du monstre diminue de 10, et il est imobilisé

pendant 1 seconde, grâce à un time.Sleep()

Les fonctions verif\_element\_fort(), verif\_element\_faible(),

verif\_nom() fonctionne de la même manière que la fonction

verif\_classe() expliquée ci dessus, en paramètre un string ou

plusieurs, et utilise des conditions qui par contre contrairement

à la fonction verif\_classe() utilise le booléen, et return true si la

condition est respectée et renvoie false si la condition n'est

pas respectée.

Les fonctions classe, Archer(), Mage() et Epéiste() fonctionne de la

même manière, utilisant la structure Personnage, chacune initialise

les infos des classes, par exemple les points de vie, leur monnaie de base

etc.. en utilisant des tableau de string, des strings ou des int.

La fonction ChatTurn(), prenant en paramètre la structure Monstre

et utilisant la structure Personnage, on créer 2 variables string qu’on

appelle choix\_inventaire et choix\_menu, on utilise d’abord choix\_menu

que l’on va scanner avec fmt.Scan() dans le fmt.Print()

du menu du combat, ensuite grâce a un switch case, selon la case on choisit si la case

est égale à 1 on appel le sous menu Menu\_attaque() pour que le personnage

attaque le monstre, ensuite on enlève les dégâts grâce aux p.points\_attaque du personnage

qui enlève au point\_de\_vie\_actuel les dégâts. Dans la case égale à 2 on utilise la variable

string choix\_inventaire et on va la scanner pour savoir ce que le joueur souhaite utiliser

entre une Potion de vie et une Potion de poison, on vérifie si le personnage à assez

de place dans son inventaire pour stocker la potion choisie, grâce a un

p.verif\_espace(choix\_inventaire), ensuite on creér une condition si le choix\_inventaire

après le fmt.Scan() est égal à 1 on utilise la fonction TakePot() et on retire directement la potion

de l’inventaire grâce à un p.removeInventory dans le cas ou fmt.Scan() est égal à 2

on utilise la Potion de poison et applique les dégâts qu’elle fait au monstre et on la retire

de l’inventaire.

La fonction trainingFight(), permet lancer le combat, on initialise 3 variable, le monstre, le nombre de

tours et un compteur de morts, on compare les 2 initiatives grâce à une condition if, celui

qui a la plus grande commence, on fait une boucle for p.point\_de\_vie\_actuel >= 0 ||

m1.point\_de\_vie\_actuel > 0 pour que si le personnage ou le monstre sont négatif en point

de vie on lance pas la boucle , il faut qu’ils soient positif en point de vie les 2

on créer un système avec une condition if, qui a comme condition que si le nb de tours est

paire c’est au monstre de d’attaquer et si c’est impair c’est au personnage d’attaquer

dans le les 2 cas on ajoute 1 au nbtours pour que ce soit chacun son tour

ensuite on affiche avec un fmt.Println() le nombre de tour auquel nous sommes

ensuite on fait une condition si les points de vie du personnage sont négatifs alors

on arrête la boucle et la fonction avec un break et pareil pour le monstre

on créer une dernière condition dans lequelle les points de vie du personnage sont

négatif et vérifiés et on ajoute de l’expèrience si la condition est respectée

avec p.addExp et on créer une boule for dans laquelle on ajoute a l’inventaire ce que le

monstre avait sur lui

La fonction Equipement, d’abord nous avons créer une structure Equipement

Prenant comme attribut tout l’équipement du personnage, comme par exemple

Le casque en string, le plastron en string etc..

Pour la fonction Init\_List, dans chacun des attribut étant des tableau de string

Nous ajoutons tout les types de cet équipement possible.

Ensuite nous avons créer de nombreuses fonction pour équipé le personnage avec

L’item choisi, par exemple pour le casque, selon la qualité du casque, grâce a un switch

Case nous avons créer des différence entre les casque, certains donne plus que d’autre et

Permettent au personnage d’augmenter son nombre de point de vie maximum.

La fonction addExp, permet de créer des niveaux et de l’expérience dans le jeu

Prenant en paramètre un int, on ajoute ce nb d’expérience en plus de l’expérience

Actuel qui est un attribut dans la structure personnage. Ensuite on fait une condition

if p.experience\_actuel > p.experience\_max

pour ne pas dépasser le nombre d’experience maximum pour le personnage sur le

niveau.

La fonction addLevel(), utilise la structure Personnage et change les données des

Attributs, elle permet d’ajouter des stats pour chaque level passé, en

augmenter l’expérience, le niveau du personnage etc..

La fonction Menu\_Forgeron(), comme chaque sous menu précédemment vu, on utilise

Un fmtPrintln(), suivi d’un fmtScan() avec un switch case, dans lequel la case 1

Appel le menu Menu\_Equipement\_Creation(), la case 2 dans laquelle on créer

Une condition de vérification pour vérifier chacune des classes, et lancer selon la

Classe le menu lui étant attribué.

En accédant aux sous menus, dans la même optique, on a un fmt.Print() qui permet

A l’utilisateur de choisir spécifiquement sa demande et accéder a d’autre sous menu,

Il faut que l’utilisateur possède assez de monnaie d’où le besoin d’une fonction

Creation\_Objet() dans laquelle il y a des conditions permettant de vérifier

Si le personnage a assez de monnaie pour pouvoir s’offrir la création du forgeron

Et dans le cas ou il le peut, on lui retire de son inventaire le coût grâce à un

p.removeInventory et on ajoute l’item dans l’inventaire grâce à un p.addInventory.

La fonction CheckInventory() qui marche avec un booléen pour s’assurer que la taille

De l’inventaire len(p.inventaire), permet en cas d’appel de vérifier que l’inventaire

Du personnage peut recevoir un item, qu’il reste de la place.

Les fonctions addInventory() et removeInventory(), permettent l’ajout ou le retrait

D’item dans l’inventaire, la fonction addInventory() prend en argument

Un string, on a une condition if qui vérifie que l’inventaire à de la place

Pour pouvoir recevoir un item :

if len(p.inventaire) < p.taille\_inventaire

Si la condition est respectée on ajoute l’item.

La fonction removeInventory(), prenant en argument un string qu’on appelle s

quant-à elle fonctionne avec une boucle for qui initialise i vaut 0, sa

condition d’arrêt est d’être plus petit que la taille de l’inventaire, ensuite on a

une condition if :

if p.inventaire[i] == s

Si la condition est respectée, on supprime l’élément de la liste grâce à :

p.inventaire = append(p.inventaire[:i], p.inventaire[i+1:]...)

La fonction Monstre, on créer d’abord la structure Monstre avec des attributs

Qui le définisse, ensuite on créer la fonction InitMonstre() avec 3 conditions

Chacune vérifiant l’argument de la fonction en string, si ils sont égaux

Alors on initialise le monstre avec tout ses attributs et les changements.