Uitleg Dungeon Game Code

**Includen van libraries**

1. #include <stdlib.h> // Voor functies als malloc(), free(), rand(), etc.

2. #include <stdio.h> // Voor printf(), scanf(), etc.

3. #include <string.h> // Voor functies als strcpy(), strcmp(), etc.

4. #include <time.h> // Voor tijdsfuncties, bv. voor srand(time(NULL))

**Constantes voor de monsters**

1. #define BABY\_DRAGON\_HEALTH 200

2. #define ANCIENT\_DRAGON\_HEALTH 400

3. #define BABY\_DRAGON\_DAMAGE 2

4. #define ANCIENT\_DRAGON\_DAMAGE 5

Hier worden **constante waarden** ingesteld voor de baby-draak en oude draak (levenspunten en schade).  
– Zo kan je deze waarden makkelijk aanpassen zonder de hele code door te zoeken.

**Struct voor kamerverbindingen**

1. typedef struct RoomLink {

2. struct Room\* connected\_room;

3. struct RoomLink\* next;

4. } RoomLink;

Dit is een **linked list** voor de deuren van een kamer.  
Elke kamer kan meerdere verbindingen hebben met andere kamers (connected\_room).  
next wijst naar de volgende link (deur) in de lijst.

**Struct voor kamers**

1. typedef struct Room {

2. int id; // Uniek id van de kamer

3. int door\_count; // Aantal deuren in de kamer

4. int has\_monster; // 0 = geen, 1 = baby draak, 2 = oude draak

5. int searched; // Kamer doorzocht? (1) of niet (0)

6. \_Bool visited; // Kamer bezocht? true/false

7. \_Bool cleared; // Kamer leeg (na verslaan monster)? true/false

8. \_Bool has\_treasure; // Schat aanwezig? true/false

9. RoomLink\* links; // Lijst van verbonden kamers

10. } Room;

Room stelt één kamer voor in de dungeon.  
– Houdt bij of er monsters, schatten of andere eigenschappen zijn.  
– links wijst naar de deuren/verbindingen.

**Struct voor de dungeon**

1. typedef struct {

2. int room\_count; // Aantal kamers

3. Room\* rooms; // Array van kamers

4. } Dungeon;

Dungeon bevat alle kamers van het spel.  
– rooms is een pointer naar de array van alle kamers.

**Globale variabelen voor het spel**

1. int monster\_attack = 0;

2. int player\_attack = 0;

3. int random\_num = 0;

4. int game\_ended = 0;

Variabelen die je overal in het programma kunt gebruiken.  
– Bijvoorbeeld: bijhouden wie aanvalt en met hoeveel schade.

**Struct voor de speler**

1. typedef struct {

2. char name[50];

3. int health;

4. int attack\_power;

5. int current\_room;

6. } Player;

Gegevens van de speler zelf: naam, gezondheid, kracht en locatie (welke kamer).

**Struct voor een monster**

1. typedef struct {

2. char\* name;

3. int health;

4. int attack\_power;

5. } Monster;

Standaardstruct om **monsters** te definiëren.  
– Kan gebruikt worden voor baby en oude draak.

**De variabelen voor de speler en monsters**

1. Player player;

2. Monster baby\_dragon;

3. Monster ancient\_dragon;

De globale instanties van speler en monsters.  
– Deze worden later in de code gevuld (bijv. in main() of een init-functie).

**Functie-pointers voor speciale acties**

1. typedef void (\*MonsterGenerator)(Room\*);

2. typedef void (\*BattleHandler)(Monster\*, Room\*);

3. typedef void (\*ItemHandler)(Room\*);

Dit zijn **typedefs** voor functie-pointers:

* **MonsterGenerator**: wijst naar een functie die een monster genereert in een kamer.
* **BattleHandler**: wijst naar een functie die een gevecht start tussen speler en monster.
* **ItemHandler**: wijst naar een functie die een item-effect uitvoert in een kamer.

1. void create\_monsters(Room \*room);

2. void start\_battle(Monster \*monster, Room \*room);

Dit zijn de functies die deze pointers later gebruiken.

1. MonsterGenerator generate\_monster = &create\_monsters;

2. BattleHandler fight = &start\_battle;

Hier worden de functie-pointers ingesteld.

**Item-acties**

* **health\_potion()**: verhoogt de HP van de speler met een random waarde tussen 20 en 50.
* **power\_up()**: verdubbelt de aanvalskracht van de speler tijdelijk.
* **trap()**: doet schade aan de speler tussen 20 en 50 HP.

1. ItemHandler item\_actions[3] = {

2. &health\_potion,

3. &power\_up,

4. &trap

5. };

Een array van de drie item-acties, zodat je later random een item kan kiezen.

**Monster generatie**

1. void create\_monsters(Room \*room){

2. if ((rand() % 4) == 0){

3. if ((rand() % 3) == 0){

4. room->has\_monster = 2; // Oude Draak

5. }

6. else{

7. room->has\_monster = 1; // Baby Draak

8. }

9. }

10. else{

11. room->has\_monster = 0; // Geen monster

12. }

13. }

Elke kamer heeft een kans (1 op 4) om een monster te krijgen. Kans op oude draak is kleiner.

**Kamers met elkaar verbinden**

1. // Kamers verbinden

2. void link\_rooms(Room\* room1, Room\* room2){

3.     // Voorkom dubbele verbinding

4.     RoomLink\* current = room1->links;

5.     while (current != NULL) {

6.         if (current->connected\_room == room2) return;

7.         current = current->next;

8.     }

9.

10.     RoomLink\* link1 = malloc(sizeof(RoomLink));

11.     link1->connected\_room = room2;

12.     link1->next = room1->links;

13.     room1->links = link1;

14.     room1->door\_count++;

15.

16.     RoomLink\* link2 = malloc(sizeof(RoomLink));

17.     link2->connected\_room = room1;

18.     link2->next = room2->links;

19.     room2->links = link2;

20.     room2->door\_count++;

21. }

Deze functie maakt een tweezijdige link tussen twee kamers (geen dubbele links).

**Beschikbare kamer zoeken**

1. int find\_available\_room(Dungeon\* dungeon, int current\_id){

2. // Zoek kamers met minder dan 4 deuren

3. }

Vind een kamer die nog niet vol zit met verbindingen.

**Dungeon opbouwen**

1. Dungeon\* build\_dungeon(int room\_count){

2. // Maak de dungeon aan

3. // Stel kamers in, voeg monsters toe

4. // Wijs een kamer aan als schatkamer

5. // Verbind kamers

6. }

Bouwt de hele dungeon op, vult de kamers, en zet de schatkamer.

**Dungeon opruimen**

1. void destroy\_dungeon(Dungeon\* dungeon){

2. // Ruim alle links en kamers op

3. }

Voorkomt geheugenlekken na het spel.

**Binair vechtsysteem**

1. void binary\_fight(){

2. // Zet random\_num om in binaire cijfers

3. // Elk bit bepaalt of monster of speler aanvalt

4. }

Elke bit in random\_num (max 16 bits) bepaalt of de speler of monster aanvalt in een gevechtsronde.

**Items verwerken**

1. void process\_items(Room \*current\_room) {

2. // Controleer of kamer nog niet doorzocht is

3. // Mogelijkheid op schat vinden of een item activeren

4. }

Verwerk wat er in de kamer ligt (schat, items).

**Gevechtsacties**

1. void player\_attack\_action(Monster \*m, Room \*r) { ... }

2. void monster\_attack\_action(Monster \*m, Room \*r) { ... }

Schade berekenen voor speler en monster.

1. void start\_battle(Monster \*m, Room \*current\_room){

2. // Gevecht in een loop totdat iemand sterft

3. // Elke ronde: binary\_fight()

4. // Check of speler of monster dood is

5. }

Handler voor het gevecht.

**Spel opslaan en laden**

1. void save\_game(Dungeon \*d, Player \*p) { ... }

2. int load\_game(Dungeon \*\*d, Player \*p) { ... }

Hier bewaar en laad je de dungeon en speler-gegevens.

**Opzetten van de random generator**

1. srand(time(NULL));

Initialiseer de random generator op basis van de huidige tijd. Dit maakt de willekeurige elementen in het spel (zoals dungeon lay-out of events) uniek bij elke run.

**Variabelen en dungeon initialiseren**

1. Dungeon\* dungeon = NULL;

Pointers voor de dungeonstructuur, begint op NULL.

**Checken of er een opgeslagen spel is**

1. FILE \*save\_file = fopen("save.dat", "rb");

2. if (save\_file) {

3. fclose(save\_file);

4. printf("Opgeslagen spel gevonden. Laden? (1=Ja, 0=Nee): ");

5. ...

6. }

Als er een save-bestand is (save.dat), vraag de speler of die wil laden.

Bij keuze '1' wordt load\_game() aangeroepen. Als dit faalt, start het spel opnieuw.

**Nieuw spel starten als er geen opgeslagen spel is**

1. if (!dungeon) {

2. printf("Voer je naam in: ");

3. fgets(player.name, sizeof(player.name), stdin);

4. player.name[strcspn(player.name, "\n")] = '\0';

Spelersnaam vragen. De strcspn zorgt dat de newline (\n) wordt verwijderd.

Vervolgens wordt de speler verwelkomd.

**Speler instellen**

1. player.health = 100;

2. player.attack\_power = 5;

Speler krijgt begin-levenspunten en aanvalskracht.

**Dungeon aanmaken**

1. int room\_count;

2. printf("Hoeveel kamers wil je? (2-25): ");

3. scanf("%d", &room\_count);

4. while(getchar() != '\n');

5.

6. if(room\_count < 2 || room\_count > 25){

7. printf("Ongeldig aantal. Standaardwaarde 10 wordt gebruikt.\n");

8. room\_count = 10;

9. }

10. dungeon = build\_dungeon(room\_count);

11. player.current\_room = 0;

Speler kiest aantal kamers (minimaal 2, maximaal 25).

Als het getal buiten deze range valt, standaard 10 kamers.

build\_dungeon maakt de dungeon aan, speler start in kamer 0.

**Monsters instellen**

1. baby\_dragon.name = "Baby Draak";

2. baby\_dragon.health = BABY\_DRAGON\_HEALTH;

3. baby\_dragon.attack\_power = BABY\_DRAGON\_DAMAGE;

4.

5. ancient\_dragon.name = "Oude Draak";

6. ancient\_dragon.health = ANCIENT\_DRAGON\_HEALTH;

7. ancient\_dragon.attack\_power = ANCIENT\_DRAGON\_DAMAGE;

De monsters (baby draak en oude draak) worden ingesteld met hun eigenschappen.

**Hoofdlus van het spel**

1. while(!game\_ended){

2. ...

3. }

Het spel loopt zolang game\_ended niet waar is.

**Kamerstatus**

1. Room\* current\_room = &dungeon->rooms[player.current\_room];

2. current\_room->visited = 1;

3.

4. printf("Je bent in kamer %d\n", current\_room->id);

Huidige kamer ophalen en markeren als bezocht.

**Verbindingen tonen**

1. if(current\_room->cleared){

2. printf("Verbindingen: ");

3. RoomLink\* link = current\_room->links;

4. while(link != NULL){

5. printf("%d ", link->connected\_room->id);

6. link = link->next;

7. }

8. }

Als de kamer al is vrijgemaakt (cleared), toon de verbindingen naar andere kamers.

**Items verwerken**

1. process\_items(current\_room);

2. if(game\_ended == 1){

3. destroy\_dungeon(dungeon);

4. return 0;

5. }

Kijkt of er items in de kamer liggen (process\_items). Mogelijk eindigt het spel hier.

**Gevecht met monsters**

1. if(!current\_room->cleared && current\_room->has\_monster){

2. switch(current\_room->has\_monster){

3. case 1:

4. baby\_dragon.health = BABY\_DRAGON\_HEALTH;

5. (\*fight)(&baby\_dragon, current\_room);

6. break;

7. case 2:

8. ancient\_dragon.health = ANCIENT\_DRAGON\_HEALTH;

9. (\*fight)(&ancient\_dragon, current\_room);

10. break;

11. }

12. if(game\_ended == 1){

13. break;

14. }

15. }

Als de kamer nog niet vrij is en er is een monster, start het gevecht.

De fight-functie pointer wordt gebruikt om de juiste gevechtslogica aan te roepen.

Als het spel hierna afgelopen is (game\_ended), breek de loop.

**Speler laten kiezen voor een volgende kamer of opslaan**

1. printf("\nBeschikbare kamers: ");

2. RoomLink\* link = current\_room->links;

3. while(link != NULL){

4. printf("%d ", link->connected\_room->id);

5. link = link->next;

6. }

7.

8. char input[10];

9. printf("\nKies een kamer (-1 om te stoppen, 'save' om op te slaan): ");

10. fgets(input, sizeof(input), stdin);

11. input[strcspn(input, "\n")] = '\0';

Toon beschikbare kamers (verbonden kamers).

Speler kiest een kamer, of ‘save’ om op te slaan.

**Opslaan of naar een nieuwe kamer gaan**

1. if(strcmp(input, "save") == 0) {

2. save\_game(dungeon, &player);

3. printf("Spel opgeslagen!\n");

4. continue;

5. }

Als de speler ‘save’ typt, sla het spel op.

**Speler verplaatsen naar nieuwe kamer**

1. int new\_room = atoi(input);

2. if(new\_room == -1) break;

3.

4. \_Bool valid = 0;

5. link = current\_room->links;

6. while(link && !valid){

7. if(link->connected\_room->id == new\_room){

8. printf("%s verlaat kamer %d en gaat naar kamer %d\n", player.name, player.current\_room, new\_room);

9. valid = 1;

10. player.current\_room = new\_room;

11. }

12. link = link->next;

13. }

14.

15. if(!valid){

16. printf("Ongeldige keuze!\n");

17. }

Check of de gekozen kamer geldig is en verbind met huidige kamer.

Verplaats de speler daarheen, of geef een foutmelding bij een ongeldig nummer.

**Dungeon opruimen na afloop**

1. destroy\_dungeon(dungeon);

2. return 0;

Na het spel wordt het geheugen opgeruimd (destroy\_dungeon).