RPG DB C++ Internals

1 Copyright notice.

Role Playing DB – A database package that creates and maintains a database of RPG characters, monsters, treasures, spells, and playing environments.

2 Introduction.

This manual describes the C++ class library used by the Role Playing Database. This consists of a collection of classes for the various base data structures used to represent the various informational objects used in Role Playing Games. These informational objects consist of *Characters*, *Monsters*, *"Magic" Spells*, *(Dungeon) Dressings*, and *"Spaces"*. All of these data objects are built upon a common low-level data object, a `Record`. Also included is a specialized random number generator that emulates one or more dice, dice being commonly used to implement random `chance` or "fate" in most Role Playing Games[1].

Although TSR's *Advanced Dungeons & Dragons* [1], [2], and [3] were used as a reference in the design of these data structures, they should be generic enough to be usable for any other Role Playing Game system.

[1] Some use shuffled decks of cards and others use a mix.

3 Random number games.

Names

First a PD random number generator then a class that implements dice.

3.1 class RandomInteger

Public Members

Private Members

RandomInteger class – generate a random number in the range specified.

This random number generator originally appeared in "Toward a Universal Random Number Generator" by George Marsaglia and Arif Zaman. Florida State University Report: FSU-SCRI-87-50 (1987)

It was later modified by F. James and published in "A Review of Pseudo-random Number Generators".

Converted from FORTRAN to C by Phil Linttell, James F. Hickling Management Consultants Ltd. Aug. 14, 1989.

THIS IS THE BEST KNOWN RANDOM NUMBER GENERATOR AVAILABLE. (However, a newly discovered technique can yield a period of 10^{600}. But that is still in the development stage.)

It passes ALL of the tests for random number generators and has a period of 2^{144}, is completely portable (gives bit identical results on all machines with at least 24-bit mantissas in the floating point representation).

The algorithm is a combination of a Fibonacci sequence (with lags of 97 and 33, and operation "subtraction plus one, modulo one") and an "arithmetic sequence" (using subtraction).

On a Vax 11/780, this random number generator can produce a number in 13 microseconds.

3.1.4

RandomInteger (int low,int high)

constructor;

3.1.1

int **l**

lower bound.

3.1.2

int **h**

upper bound.

3.1.3

int **rmarin** (int ij, int kl)

This is the initialization routine for the random number generator RANMAR(). NOTE: The seed variables can have values between: $0 \leq IJ \leq 31328$ $0 \leq KL \leq 30081$. The random number sequences created by these two seeds are of sufficient length to complete an entire calculation with. For example, if

several different groups are working on different parts of the same calculation, each group could be assigned its own IJ seed. This would leave each group with 30000 choices for the second seed. That is to say, this random number generator can create 900 million different subsequences – with each subsequence having a length of approximately 10^{30}.

Use $IJ = 1802$ and $KL = 9373$ to test the random number generator. The subroutine RANMAR should be used to generate 20000 random numbers. Then display the next six random numbers generated multiplied by 4096*4096. If the random number generator is working properly, the random numbers should be:

6533892.0 14220222.0 7275067.0
6172232.0 8354498.0 10633180.0

3.2 class **Dice**

Public Members

Private Members

Dice Class – Just a random number generator using a dice model.

3.2.5

Dice (unsigned int ns, unsigned int nd = 1)

Arguments

3.2.5.1 unsigned int
ns 23

3.2.5.2 unsigned int
nd 23

Constructor, create 1 or more dice.

3.2.5.1

unsigned int **ns**

Number of sides (0 = percentile dice).

3.2.5.2

unsigned int **nd**

Number of dice.

3.2.6

Dice ()

Constructor – uninitialized dice.

3.2.7

~Dice ()

Destructor.

3.2.8

unsigned int **Roll** ()

Roll them bones...

3.2.9

bool **TypeOfDice** (unsigned int& ns, unsigned int& nd)

Arguments

Return the type of dice.

3.2.9.1

unsigned int& **ns**

Number of sides (0 = percentile dice).

3.2.9.2

unsigned int& **nd**

Number of dice.

3.2.1

RandomInteger * **dice**

Random number generator.

3.2.2

unsigned int **nsides**

Number of sides (0 = percentile dice).

3.2.3

unsigned int **ndice**

Number of dice.

3.2.4

void **~Dice** ()

Internal initializer.

4

struct **Record**

Public Members

Record structure.

Core resident record - it has a size and a chunk of memory (the record itself). This structure is lifted from the Home Librarian package. A very re-usable piece of code.

Lots of fun. I wish C++ had a garbage collector...

4.1

size_t **size**

Size of the record.

4.2

char * **buffer**

Buffer for the data.

4.3

Record ()

Base constructor: empty buffer.

4.4

Record (size_t size)

Constructor: preallocated buffer.

4.5

Record (const Record& rec)

Constructor: record from another record.

4.6

~Record ()

Destructor: free up allocated memory.

4.7

Record& **operator =** (const Record& rec)

Assignment operator (copy a record) – allocate fresh memory.

4.8

void **NewBuffer** (size_t size)

Buffer (re-)allocator function.

5

class **Character**

Public Members

Private Members

Basic character class. Contains all of the information needed to describe a player or non-player character.

5.28

Character ()

Base constructor.

5.29

Character (int s, int i, int w, int d, int c, int ch, int es = 0, int l = 1, int hd = 6, int nd = 1, int md = 1, const char *n = "", const char *p = "", const char *r = "Human", const char *chc = "npc", const char *align = "", const char *sex = "", int ag = 0, const char *com = "", const char *im = "")

Arguments

Fighter character constructor (extra strength parameter).

5.29.1

int **s**

Basic strength.

5.29.2

int **i**

Intelligence.

5.29.3

int **w**

Wisdom.

5.29.4

int **d**

Dexterity.

5.29.5

int **c**

Constitution.

5.29.6

int **ch**

Charisma.

5.29.7

int **es**

Extra strength.

5.29.8

int **l**

Level.

5.29.9

int **hd**

Hit die.

5.29.10

int **nd**

Number of hit dice.

5.29.11

int **md**

Maximum number of hit dice.

5.29.12

const char * **n**

Character's name.

5.29.13

const char * **p**

Player's name.

5.29.14

const char * **r**

Character's race.

5.29.15

const char * **chc**

Character's class.

5.29.16

const char * **align**

Character's alignment.

5.29.17

const char * **sex**

Character's sex.

5.29.18

int **ag**

Character's age.

5.29.19

const char * **com**

Commentary.

5.29.20

const char * **im**

Image.

5.30

Character (int s, int i, int w, int d, int c, int ch, int l = 1, int hd = 6, int nd = 1, int md = 1, const char *n = "", const char *p = "", const char *r = "Human", const char *chc = "npc", const char *align = "", const char *sex = "", int ag = 0, const char *com = "", const char *im = "")

Arguments

Non Fighter character constructor (no extra strength parameter).

5.30.1

int **s**

Basic strength.

5.30.2

int **i**

Intelligence.

5.30.3

int **w**

Wisdom.

5.30.4

int **d**

Dexterity.

5.30.5

int **c**

Constitution.

5.30.6

int **ch**

Charisma.

5.30.7

int **l**

Level.

5.30.8

int **hd**

Hit die.

5.30.9

int **nd**

Number of hit dice.

5.30.10

int **md**

Maximum number of hit dice.

5.30.11

const char * **n**

Character's name.

5.30.12

const char * **p**

Player's name.

5.30.13

const char * **r**

Character's race.

5.30.14

const char * **chc**

Character's class.

5.30.15

const char * **align**

Character's alignment.

5.30.16

const char * **sex**

Character's sex.

5.30.17

int **ag**

Character's age.

5.30.18

const char * **com**

Commentary.

5.30.19

const char * **im**

Image.

5.31

Character (const Character *that)

Copy constructor (from pointer).

5.32

Character (const Character &that)

Copy constructor (from reference).

5.33

Character (const Record *rec)

Type conversion constructor, from a Record.

5.34

~Character ()

Destructor.

5.35

operator const **Record** () const

Type conversion: convert to a Record.

5.36

int **HitPoints** () const

Return hit points.

5.37

int **Age** () const

Return age.

5.38

int **Strength** () const

Return strength.

5.39

int **Intelligence** () const

Return intelligence.

5.40

int **Wisdom** () const

Return wisdom.

5.41

int **Dexterity** () const

Return dexterity.

5.42

int **Constitution** () const

Return constitution.

5.43

int **Charisma** () const

Return charisma.

5.44

int **ExceptionalStrength** () const

Return exceptional strength.

5.45

int **Level** () const

Return level.

5.46

int **ExperiencePoints** () const

Return experience points.

5.47

int **Gold** () const

Return gold.

5.48

const char * **Name** () const

Return name.

5.49

const char * **Player** () const

Return player.

5.50

const char * **Race** () const

Return race.

5.51

const char * **CharacterClass** () const

Return character class.

5.52

const char * **Alignment** () const

Return alignment.

5.53

const char * **Sex** () const

Return sex.

5.54

const char * **Comments** () const

Return Comments.

5.55

const char * **Image** () const

Return image.

5.56

int **SetAge** (int newage)

Set age.

5.57

int **SetStrength** (int newS)

Set strength.

5.58

int **SetIntelligence** (int newI)

Set intelligence.

5.59

int **SetWisdom** (int newW)

Set wisdom.

5.60

int **SetDexterity** (int newD)

Set dexterity.

5.61

int **SetConstitution** (int newC)

Set constitution.

5.62

int **SetCharisma** (int newCh)

Set charisma.

5.63

int **SetExceptionalStrength** (int newES)

Set exceptional strength.

5.64

int **SetExperiencePoints** (int newEP)

Set experience points.

5.65

int **SetGold** (int newGP)

Set gold.

5.66

const char * **SetName** (const char * newN)

Set name.

5.67

const char * **SetPlayer** (const char * newP)

Set player.

5.68

const char * **SetRace** (const char *newR)

Set race.

5.69

const char * **SetCharacterClass** (const char *newCh)

Set character class.

5.70

const char * **SetAlignment** (const char *newA)

Set alignment.

5.71

const char * **SetSex** (const char *newS)

Set sex.

5.72

const char * **SetComments** (const char *newC)

Set comments.

5.73

const char * **SetImage** (const char *newI)

Set image.

5.74

void **AdvanceLevel** ()

Advance character's level.

5.75

void **UpdateFromRecord** (const Record &rec)

Update character from Record.

5.1

const char * **name**

Character name.

5.2

const char * **player**

Player name.

5.3

const char * **race**

Character race.

5.4

const char * **chclass**

Character class.

5.5

const char * **alignment**

Character alignment.

5.6

const char * **sex**

Character sex.

5.7

const char * **comments**

Additional comments.

5.8

const char * **image**

Character's image (GIF file).

5.9

int **hitpoints**

Character hit points.

5.10

int **age**

Character age.

5.11

int **strength**

Character strength.

5.12

int **intelligence**

Character intelligence.

5.13

int **wisdom**

Character wisdom.

5.14

int **dexterity**

Character dexterity.

5.15

int **constitution**

Character constitution.

5.16

int **charisma**

Character charisma.

5.17

int **exceptional_strength**

Character exceptional strength.

5.18

int **level**

Character level.

5.19

int **experience_points**

Experience points.

5.20

int **gold**

Gold pieces.

5.21

int **hdie**

Character hit die.

5.22

int **ndie**

Character number of hit dice.

5.23

int **maxdie**

Character maximum number of hit dice.

5.24

Record **rawData**

Data record.

5.25

void **_Character** (int s, int i, int w, int d, int c, int ch, int es, int l, int hd, int nd, int md, const char * n, const char * p, const char * r, const char * chc, const char * align, const char * sex, int ag,const char * com, const char * im)

Internal initializer function.

5.26

void **RecordToCharacter** ()

Convert internal record to character.

5.27

void **UpdateRecord** ()

Update internal record.

6

class Monster

Public Members

Private Members

Basic monster class. Contains all of the information needed to describe a monster.

6.1

enum **FreqType**

Members

Frequency Types.

6.1.1

Unique

This monster is unique (only one exists).

6.1.2

VeryRare

This monster is very rare, only a few exist. 4% probability of occurrence.

6.1.3

Rare

This monster is rare, not many exist. 11% probability of occurrence.

6.1.4 Uncommon

This monster is uncommon, some exist but are not seen often. 20% probability of occurrence.

6.1.5 Common

This monster is common, they are all over the place. 65% probability of occurrence.

6.1.6 BogusFrequency

This is a bogus frequency. Used to handle uninitialized values.

6.2 enum IntelligenceRating

Members

Intelligence Ratings.

6.2.1

Non

Unintelligent. Mindless critter. I score = 0.

6.2.2

Animal

Animal intelligence. I score = 1.

6.2.3

Semi

Semi-intelligent. I score = 2-4.

6.2.4

Low

Low intelligence. I score = 5-7.

6.2.5 Average

Average (Human) intelligence. I score = 8-10.

6.2.6 Very

Very intelligent. I score = 11-12.

6.2.7 Highly

Highly intelligent. I score = 13-14.

6.2.8 Exceptionally

Exceptionally intelligent. I score = 15-16.

6.2.9 Genius

Genius. I score = 17-18.

6.2.10 SupraGenius

Supra-Genius. I score = 19-20.

6.2.11 Godlike

Godlike intelligence. I score = 21+.

6.2.12 BogusIntelligence

This is a bogus intelligence. Used to handle uninitialized values.

6.3 enum HitType

Members

Method of hit points.

6.3.1 Points

An explicit number of hit points. Typical for unique monsters, like demigods, major dragons, or demons.

6.3.2 Dice

Normal hit dice system. Typical for monsters that are more common, like orcs.

6.3.3 BogusHitType

This is a bogus value. Used to handle uninitialized values.

6.36 const char * **Name** () const

Return the name/type.

6.37

const char * **Alignment** () const

Return the alignment.

6.38

const char * **TreasureType** () const

Return the treasure type.

6.39

const char * **SpecialAttacks** () const

Return the special attacks.

6.40

const char * **SpecialDefences** () const

Return the special defenses.

6.41

const char * **Psionics** () const

Return the psionics.

6.42

const char * **Comments** () const

Return the commentary.

6.43

const char * **Image** () const

Return the image.

6.44

int **HitDieSides** () const

Return the hit die sides.

6.45

int **NumHitDice** () const

Return the number of hit dice.

6.46

int **HitAdjustment** () const

Return the hit dice adjustment.

6.47

int **ArmorClass** () const

Return the armor class.

6.48

int **LandSpeed** () const

Return the speed on land.

6.49

int **FlyingSpeed** () const

Return the speed in the air (flying).

6.50

int **SwimmingSpeed** () const

Return the speed in the water (swimming).

6.51

int **BurrowingSpeed** () const

Return the speed in the earth (burrowing).

6.52

int **WebSpeed** () const

Return the speed in web.

6.53

int **PercentInLair** () const

Return the percent in lair.

6.54

int **NumberOfAttacks** () const

Return the number of attacks.

6.55

int **MagicalResistance** () const

Return the magical resistance.

6.56

int **HitPoints** () const

Return the number of hit points.

6.57

void **DamagePerAttack** (int& L, int& H) const

Arguments

Return the damage per attack.

6.57.1

int& **L**

Lower bound.

6.57.2

int& **H**

Upper bound.

6.58

void **NumberAppearing** (int& L, int& H) const

Arguments

Return the number appearing.

6.58.1

int& **L**

Lower bound.

6.58.2

int& **H**

Upper bound.

6.59

IntelligenceRating **Intelligence** () const

Return the intelligence rating.

6.60

FreqType **Frequency** () const

Return the frequency.

6.61

HitType **Hittype** () const

Return the hit type.

6.62

double **Size** () const

Return the size.

6.63

Monster ()

Default constructor.

6.64

Monster (int hp, int ac, int m, int mf, int ms, int mb, int mw, int pl, int na, int daL, int daH, int mr, int noaL, int noaH, IntelligenceRating i, FreqType f, double s, const char * nm, const char * al, const char * tt, const char * sa, const char * sd, const char * ps, const char * com, const char * im)

Arguments

Hit point type constructor (unique and very rare monsters).

6.64.1

int **hp**

Hit points.

6.64.2

int **ac**

Armor class.

6.64.3

int **m**

Land speed.

6.64.4

int **mf**

Flying speed.

6.64.5

int **ms**

Swimming speed.

6.64.6

int **mb**

Burrowing speed.

6.64.7

int **mw**

Speed in web.

6.64.8

int **pl**

Percent in lair.

6.64.9

int **na**

Number of attacks.

6.64.10

int **daL**

Minimum damage per attack.

6.64.11

int **daH**

Maximum damage per attack.

6.64.12

int **mr**

Magical resistance.

6.64.13

int **noaL**

Minimum number appearing.

6.64.14

int **noaH**

Maximum number appearing.

6.64.15

IntelligenceRating **i**

Intelligence rating.

6.64.16

FreqType **f**

Frequency type.

6.64.17

double **s**

Size.

6.64.18

const char * **nm**

Name/type.

6.64.19

const char * **al**

Alignment.

6.64.20

const char * **tt**

Treasure type.

6.64.21

const char * **sa**

Special attacks.

6.64.22

const char * **sd**

Special defenses.

6.64.23

const char * **ps**

Psionics.

6.64.24

const char * **com**

Commentary.

6.64.25

const char * **im**

Image.

6.65

Monster (int hd, int nd, int had, int ac, int m, int mf, int ms, int mb, int mw, int pl, int na, int daL, int daH, int mr, int noaL, int noaH, IntelligenceRating i, FreqType f, double s, const char * nm, const char * al, const char * tt, const char * sa, const char * sd, const char * ps, const char * com, const char * im)

Arguments

Hit dice type constructor (more common monsters).

6.65.1

int **hd**

Hit dice type (usually 8).

6.65.2

int **nd**

Number of hit dice.

6.65.3

int **had**

Hit point adjustment.

6.65.4

int **ac**

Armor class.

6.65.5

int **m**

Land speed.

6.65.6

int **mf**

Flying speed.

6.65.7

int **ms**

Swimming speed.

6.65.8

int **mb**

Burrowing speed.

6.65.9

int **mw**

Speed in web.

6.65.10

int **pl**

Percent in lair.

6.65.11

int **na**

Number of attacks.

6.65.12

int **daL**

Minimum damage per attack.

6.65.13

int **daH**

Maximum damage per attack.

6.65.14

int **mr**

Magical resistance.

6.65.15

int **noaL**

Minimum number appearing.

6.65.16

int **noaH**

Maximum number appearing.

6.65.17

IntelligenceRating **i**

Intelligence rating.

6.65.18

FreqType **f**

Frequency type.

6.65.19

double **s**

Size.

6.65.20

const char * **nm**

Name/type.

6.65.21

const char * **al**

Alignment.

6.65.22

const char * **tt**

Treasure type.

6.65.23

const char * **sa**

Special attacks.

6.65.24

const char * **sd**

Special defenses.

6.65.25

const char * **ps**

Psionics.

6.65.26

const char * **com**

Commentary.

6.65.27

const char * **im**

Image.

6.66

Monster (const Monster *that)

Copy constructor (from pointer).

6.67

Monster (const Monster &that)

Copy constructor (from reference).

6.68

Monster (const Record *rec)

Type conversion constructor, from a Record.

6.69

~Monster ()

Destructor.

6.70

operator const **Record** () const

Type conversion: convert to a Record.

6.71

void **UpdateFromRecord** (const Record &rec)

Update Monster from Record.

6.4

const char * **name**

Monster's (type) name.

6.5

const char * **alignment**

Monster's alignment.

6.6

const char * **treasureType**

Monster's treasure type.

6.7

const char * **specialAttacks**

Special attacks.

6.8

const char * **specialDefences**

Special defenses.

6.9

const char * **psionics**

Psionic abilities.

6.10

const char * **comments**

Other descriptive information.

6.11

const char * **image**

A picture.

6.12

short int **hitpoints**

Monster's hitpoints.

6.13

signed char **hdie**

Monster's hit die type (usually 8).

6.14

signed char **ndie**

Number of hit dice.

6.15

signed char **hplus**

Hit point adjustment.

6.16

signed char **armclass**

Armor class.

6.17

signed char **move**

Speed on land (walking/running).

6.18

signed char **move_fly**

Speed in the air (flying).

6.19

signed char **move_swim**

Speed in water (swimming).

6.20

signed char **move_burrow**

Speed in the earth (burrowing).

6.21

signed char **move_web**

Speed in web.

6.22

signed char **percentLair**

Percent in lair.

6.23

signed char **numatt**

Number of attacks.

6.24

signed char **magres**

Magical resistance.

6.25

struct **Range**

Public Members

Range structure (used for damage per attack and number appearing).

6.25.1

signed char **l**

Lower bound.

6.25.2

signed char **h**

Upper bound.

6.26

Range **damatt**

Damage per attack.

6.27

Range **noappearing**

Number appearing.

6.28

IntelligenceRating **intelligence**

Intelligence.

6.29

FreqType **frequency**

Frequency.

6.30

HitType **hittype**

Hit type.

6.31

double **size**

Size in inches.

6.32

Record **rawData**

Data record.

6.33

void **RecordToMonster** ()

Convert internal record to monster.

6.34

void **UpdateRecord** ()

Update internal record.

6.35

void **_Monster** (int hp, int hd, int nd, int hadj, int ac, int m, int mf, int ms, int mb, int mw, int pl, int na, int daL, int daH, int mr, int noaL, int noaH, IntelligenceRating i, FreqType f, HitType h, double s, const char * nm, const char * al, const char * tt, const char * sa, const char * sd, const char * ps, const char * com, const char * im)

Arguments

Initializer.

6.35.1

int **hp**

Hit points.

6.35.2

int **hd**

Hit dice.

6.35.3

int **nd**

Number of hit dice.

6.35.4

int **hadj**

Hit point adjustment.

6.35.5

int **ac**

Armor class.

6.35.6

int **m**

Land speed.

6.35.7

int **mf**

Flying Speed.

6.35.8

int **ms**

Swimming speed.

6.35.9

int **mb**

Burrowing speed.

6.35.10

int **mw**

Speed in web.

6.35.11

int **pl**

Percent in lair.

6.35.12

int **na**

Number of attacks.

6.35.13

int **daL**

Minimum damage per attack.

6.35.14

int **daH**

Maximum damage per attack.

6.35.15

int **mr**

Magical resistance.

6.35.16

int **noaL**

Minimum number appearing.

6.35.17

int **noaH**

Maximum number appearing.

6.35.18

IntelligenceRating **i**

Intelligence rating.

6.35.19

FreqType **f**

Frequency type.

6.35.20

HitType **h**

Hit type.

6.35.21

double **s**

Size.

6.35.22

const char * **nm**

Name/Type.

6.35.23

const char * **al**

Alignment.

6.35.24

const char * **tt**

Treasure type.

6.35.25

const char * **sa**

Special attacks.

6.35.26

const char * **sd**

Special defenses.

6.35.27

const char * **ps**

Psionics.

6.35.28

const char * **com**

Commentary.

6.35.29

const char * **im**

Image.

7 class **MonsterInstance**

Public Members

Private Members

Monster "instance". Used to implement a working instance of a monster.

7.4 **MonsterInstance** (const Monster *iOf, const char *iN = "")

Constructor.

7.5

MonsterInstance (const MonsterInstance *that)

Copy constructor (from pointer).

7.6

MonsterInstance (const MonsterInstance &that)

Copy constructor (from reference).

7.7

~**MonsterInstance** ()

Destructor.

7.8

const char * **InstanceName** () const

Return the name.

7.9

int **InstanceHitPoints** () const

Return the hit points.

7.10

int **UpdateInstanceHitPoints** (int adj)

Adjust the hit points.

7.11

const Monster * **InstanceOf** () const

Return base monster object.

7.1

const char * **instanceName**

Instance's name.

7.2

const Monster * **instanceOf**

What it is an instance of.

7.3

int **instanceHitPoints**

Instance's hit points.

8

class **Spell**

Public Members

Private Members

Basic spell class. Contains all of the information needed to describe a spell.

8.17

void **UpdateFromRecord** (const Record &rec)

Update Spell from Record.

8.18

Spell (const char *spc = "", const char *nm = "", const char *spt = "", const char *desc = "", const char *a = "", const char *ct = "", const char *dt = "", const char *st = "", int l = 0, int r = 0, bool revP = false, bool verbP = false, bool somP = false, bool matP = false)

Arguments

Constructor.

8.18.1

const char * **spc**

Spell class.

8.18.2

const char * **nm**

Name.

8.18.3

const char * **spt**

Type.

8.18.4

const char * **desc**

Description.

8.18.5

const char * **a**

Effect area.

8.18.6

const char * **ct**

Casting time.

8.18.7

const char * **dt**

Duration.

8.18.8

const char * **st**

Saving throw.

8.18.9

int **l**

Level.

8.18.10

int **r**

Range.

8.18.11

bool **revP**

Reversible?

8.18.12

bool **verbP**

Verbal?

8.18.13

bool **somP**

Somatic?

8.18.14

bool **matP**

Material?

8.19

Spell (const Spell *that)

Copy constructor (from pointer).

8.20

Spell (const Spell &that)

Copy constructor (from reference).

8.21

Spell (const Record *rec)

Type conversion constructor, from a Record.

8.22

~Spell ()

Destructor.

8.23

operator const **Record** () const

Type conversion: convert to a Record.

8.24

const char * **SpellClass** () const

Return the spell's class.

8.25

const char * **Name** () const

Return the spell's name.

8.26

const char * **SpellType** () const

Return the spell's type.

8.27

const char * **Description** () const

Return the spell's description.

8.28

const char * **AreaOfEffect** () const

Return the spell's area of effect.

8.29

const char * **CastingTime** () const

Return the spell's casting time.

8.30

const char * **Duration** () const

Return the spell's duration.

8.31

const char * **SavingThrow** () const

Return the spell's Saving Throw.

8.32

int **Level** () const

Return the spell's level.

8.33

int **Range** () const

Return the spell's range.

8.34

bool **ReversibleP** () const

Return spell's reversibility flag.

8.35

bool **VerbalP** () const

Return spell's verbal component flag.

8.36

bool **SomaticP** () const

Return spell's somatic component flag.

8.37

bool **MaterialP** () const

Return spell's material component flag.

8.1

const char * **spclass**

Spell class (type of character that can use it).

8.2

const char * **name**

Spell name.

8.3

const char * **sptype**

Spell type.

8.4

const char * **description**

Descriptive text.

8.5

const char * **area**

Area of effect.

8.6

const char * **casttime**

Casting time.

8.7

const char * **duration**

Duration.

8.8

const char * **savethrow**

Saving throw.

8.9

int **level**

Spell level.

8.10

int **range**

Range.

8.11

struct **FlagBits**

Public Members

Flag bit structure.

8.11.1

unsigned **reversibile:1**

Is spell reversible?

8.11.2

unsigned **verbal:1**

Does spell use a verbal component?

8.11.3

unsigned **somatic:1**

Does spell use a somatic component?

8.11.4

unsigned **material:1**

Does spell use a material component?

8.11.5

unsigned **fill:4**

Fill out the byte.

8.12

union **FlagUnion**

Members

Flag union.

8.12.1

FlagBits **bits**

As bits.

8.12.2

unsigned char **byte**

As a byte.

8.12.3

FlagUnion ()

Constructor.

8.13

FlagUnion **flags**

Flags – components and reversibility.

8.14

Record **rawData**

Data record.

8.15

void **RecordToSpell** ()

Convert internal record to monster.

8.16

void **UpdateRecord** ()

Update internal record.

9 Dressings

Names

Various dressing classes.

9.1 class Treasure

Public Members

Private Members

Treasures are the things the characters are trying to get and the monsters are guarding.

9.1.23

void **UpdateFromRecord** (const Record &rec)

Update Treasure from Record.

9.1.24

Treasure (const char *nm = "", const char *d = "", const char *i = "", int w = 0, int aCA = 0, int toHA = 0, int damA = 0, int magResA = 0, int damProtA = 0, int sA = 0, int iA = 0, int wA = 0, int dA = 0, int cA = 0, int chA = 0, int gA = 0, int fA = 0, int swA = 0, int v = 0)

Arguments

Constructor.

9.1.24.1

const char * **nm**

What it is called.

9.1.24.2

const char * **d**

Description.

9.1.24.3

const char * **i**

Graphical image (GIF).

9.1.24.4

int **w**

How heavy it is.

9.1.24.5

int **aCA**

Armor class adjustment.

9.1.24.6

int **toHA**

To hit adjustment.

9.1.24.7

int **damA**

Damage adjustment.

9.1.24.8

int **magResA**

Magical resistance adjustment.

9.1.24.9

int **damProtA**

Damage protection adjustment.

9.1.24.10

int **sA**

Strength adjustment.

9.1.24.11

int **iA**

Intelligence adjustment.

9.1.24.12

int **wA**

Wisdom adjustment.

9.1.24.13

int **dA**

Dexterity adjustment.

9.1.24.14

int **cA**

Constitution adjustment.

9.1.24.15

int **chA**

Charisma adjustment.

9.1.24.16

int **gA**

Ground movement adjustment.

9.1.24.17

int **fA**

Flying movement adjustment.

9.1.24.18

int **swA**

Swimming movement adjustment.

9.1.24.19

int **v**

Value.

9.1.25

Treasure (const Treasure *that)

Copy constructor (from pointer).

9.1.26

Treasure (const Treasure &that)

Copy constructor (from reference).

9.1.27

Treasure (const Record *rec)

Type conversion constructor, from a Record.

9.1.28

~Treasure ()

Destructor.

9.1.29

operator const **Record** () const

Type conversion: convert to a Record.

9.1.30

const char * **Name** () const

Return name.

9.1.31

const char * **Description** () const

Return Description.

9.1.32

const char * **Image** () const

Return image.

9.1.33

int **Weight** () const

Return weight.

9.1.34

int **ArmorClassAdj** () const

Return armor class Adjustment.

9.1.35

int **ToHitAdj** () const

Return to hit adjustment.

9.1.36

int **DamageAdj** () const

Return damage adjustment.

9.1.37

int **MagicalResistanceAdj** () const

Return magical resistance adjustment.

9.1.38

int **DamageProtectionAdj** () const

Return damage protection adjustment.

9.1.39

int **StrengthAdj** () const

Return strength adjustment.

9.1.40

int **IntelligenceAdj** () const

Return intelligence adjustment.

9.1.41

int **WisdomAdj** () const

Return wisdom adjustment.

9.1.42

int **DexterityAdj** () const

Return dexterity adjustment.

9.1.43

int **ConstitutionAdj** () const

Return constitution adjustment.

9.1.44

int **CharismaAdj** () const

Return charisma adjustment.

9.1.45

int **GroundMovementAdj** () const

Return ground movement adjustment.

9.1.46

int **FlyingAdj** () const

Return flying adjustment.

9.1.47

int **SwimmingAdj** () const

Return swimming adjustment.

9.1.48

int **Value** () const

Return value.

9.1.1

const char * **name**

What it is called.

9.1.2

const char * **descr**

Description.

9.1.3

const char * **image**

Graphic.

9.1.4

int **weight**

How heavy it is.

9.1.5

int **aCAdj**

Armor Class adjustment.

9.1.6

int **toHitAdj**

To hit adjustment.

9.1.7

int **damAdj**

Damage adjustment.

9.1.8

int **magResAdj**

Magical Resistance adjustment.

9.1.9

int **damProtAdj**

Damage Protection adjustment.

9.1.10

int **sAdj**

Strength adjustment.

9.1.11

int **iAdj**

Intelligence adjustment.

9.1.12

int **wAdj**

Wisdom adjustment.

9.1.13

int **dAdj**

Dexterity adjustment.

9.1.14

int **cAdj**

Constitution adjustment.

9.1.15

int **chAdj**

Charisma adjustment.

9.1.16

int **gAdj**

Ground movement adjustment.

9.1.17

int **fAdj**

Flying movement adjustment.

9.1.18

int **swAdj**

Swimming movement adjustment.

9.1.19

int **value**

Value.

9.1.20

Record **rawData**

Data record.

9.1.21

void **RecordToTreasure** ()

Record to Treasure.

9.1.22

void **UpdateRecord** ()

Update record.

9.2

class **TrickTrap**

Public Members

Private Members

Tricks and Traps are used to protect treasure and to generally keep the

players on their toes.

9.2.8

void **UpdateFromRecord** (const Record &rec)

Update TrickTrap from Record.

9.2.9

TrickTrap (const char *nm = "", const char *tt = "", const char *d = "", const char *i = "")

Arguments

Constructor.

9.2.9.1

const char * **nm**

What it is called.

9.2.9.2

const char * **tt**

What type of trick or trap.

9.2.9.3

const char * **d**

Description.

9.2.9.4

const char * **i**

Graphical image (GIF).

9.2.10

TrickTrap (const TrickTrap *that)

Copy constructor (from pointer).

9.2.11

TrickTrap (const TrickTrap &that)

Copy constructor (from reference).

9.2.12

TrickTrap (const Record *rec)

Type conversion constructor, from a Record.

9.2.13

~TrickTrap ()

Destructor.

9.2.14

operator const **Record** () const

Type conversion: convert to a Record.

9.2.15

const char * **Name** () const

Return name.

9.2.16

const char * **Type** () const

Return trick or trap type.

9.2.17

const char * **Description** () const

Return description.

9.2.18

const char * **Image** () const

Return image.

9.2.1

const char * **name**

Name of the trick or trap.

9.2.2

const char * **tttype**

Type of trick or trap.

9.2.3

const char * **descr**

Description of the trick or trap.

9.2.4

const char * **image**

Graphic of the trick or trap.

9.2.5

Record **rawData**

Data record.

9.2.6

void **RecordToTrickTrap** ()

Record to TrickTrap.

9.2.7

void **UpdateRecord** ()

Update record.

9.3

class **Dressing**

Public Members

Private Members

Random dungeon dressings – random odds and ends. Sometimes these things have value, but real treasures use the Treasure class.

9.3.7

void **UpdateFromRecord** (const Record &rec)

Update dressing from Record.

9.3.8

Dressing (const char *nm = "", const char *d = "", const char *i = "", int v = 0)

Arguments

Constructor.

9.3.8.1

const char * **nm**

What it is called.

9.3.8.2

const char * **d**

Description.

9.3.8.3

const char * **i**

Graphical image (GIF).

9.3.8.4

int **v**

Value.

9.3.9

Dressing (const Dressing *that)

Copy constructor (from pointer).

9.3.10

Dressing (const Dressing &that)

Copy constructor (from reference).

9.3.11

Dressing (const Record *rec)

Type conversion constructor, from a Record.

9.3.12

~Dressing ()

Destructor.

9.3.13

operator const **Record** () const

Type conversion: convert to a Record.

9.3.14

const char * **Name** () const

Return name.

9.3.15

const char * **Description** () const

Return description.

9.3.16

const char * **Image** () const

Return image.

9.3.17

int **Value** () const

Return Value.

9.3.1

const char * **name**

The name of the object.

9.3.2

const char * **image**

The image of the object.

9.3.3

int **value**

The value of the object.

9.3.4

Record **rawData**

Data record.

9.3.5

void **RecordToDressing** ()

Record to Dressing.

9.3.6

void **UpdateRecord** ()

Update record.

10

Spaces – Squares and Hexes, where things happen.

Names

Space class and related odds and ends.

10.1

class **GeoConstants**

Public Members

Geometric constants.

10.1.1

const float **Width = 100.0**

Space “width”.

10.1.2

const float **HexSideLength = 57.735**

Side of a hex (computed to give a width of 100).

10.1.3

const float **HexPeakHeight = 28.8675**

Height of peak above vertical sides.

10.2

class **Exit**

Public Members

Private Members

Exit points and other inter-spatial interconnection points like windows and staircases.

10.2.1

enum **ExitType**

Members

Exit types.

10.2.1.1

Doorway

Plain doorway.

10.2.1.2

Door

Plain door.

10.2.1.3

LockedDoor

Locked door.

10.2.1.4

SecretDoor

Secret door.

10.2.1.5

OnewayDoor

One way door.

10.2.1.6

TrapDoorUp

Trapdoor, up.

10.2.1.7

TrapDoorDown

Trapdoor, down.

10.2.1.8

StairsUp

Stairs, up.

10.2.1.9

StairsDown

Stairs, down.

10.2.1.10

WindowUnglazed

Window, unglazed.

10.2.1.11

WindowGlazed

Window, glazed.

10.2.1.12

Chimney

Chimney or vent.

10.2.1.13

Pit

Pit or hole.

10.2.1.14

Unspecified

Unspecified.

10.2.9

ExitType **Type** () const

Return type of exit.

10.2.10

double **XCenter** () const

Return center X coordinate.

10.2.11

double **YCenter** () const

Return center Y coordinate.

10.2.12

bool **WallAligned** () const

Return wall alignment flag.

10.2.13

const char * **Description** () const

Return description.

10.2.14

const char * **Image** () const

Return picture of exit.

10.2.15

const char * **NextSpaceIndexString** () const

Return next space index string.

10.2.16

Exit (ExitType t = Unspecified, double x = 0.0, double y = 0.0, bool wa = true, const char *d = "", const char *im = "", const char *ns = "")

Arguments

Constructor.

10.2.16.1

ExitType **t**

Exit type.

10.2.16.2

double **x**

X coordinate.

10.2.16.3

double **y**

Y coordinate.

10.2.16.4

bool **wa**

Wall alignment?

10.2.16.5

const char * **d**

Description.

10.2.16.6

const char * **im**

Picture.

10.2.16.7

const char * **ns**

Next space.

10.2.17

~Exit ()

Destructor.

10.2.18

Exit& **operator =** (const Exit& source)

Assignment operator.

10.2.2

ExitType **type**

Type of exit.

10.2.3

float **xCenter**

Center X coordinate.

10.2.4

float **yCenter**

Center Y coordinate.

10.2.5

bool **wallAligned**

Aligned with wall?

10.2.6

const char * **descr**

Description.

10.2.7

const char * **image**

Picture of exit.

10.2.8

const char * **nextSpaceIndexString**

Next space.

10.3

class **ExitVector**

Public Members

Private Members

10.3.6 int **NearestIndex** (double x, double y) const 168

Vector of Exits.

10.3.7

ExitVector ()

Constructor.

10.3.8

~ExitVector ()

Destructor.

10.3.9

void **InsertExit** (const Exit& source)

Arguments

10.3.9.1const Exit&
source 162

Insertion function.

10.3.9.1

const Exit& **source**

Exit object to insert.

10.3.10

const Exit* operator [] (int index) const

Arguments

Index operator.

10.3.10.1

int **index**

Index.

10.3.11

const Exit* **Index** (int index) const

Arguments

Index function.

10.3.11.1

int **index**

Index.

10.3.12

const Exit* **operator ()** (double x, double y) const

Arguments

Select nearest to (x,y) operator.

10.3.12.1

double **x**

X coordinate.

10.3.12.2

double **y**

Y coordinate.

10.3.13

const Exit* **Nearest** (double x, double y) const

Arguments

Select nearest to (x,y) function.

10.3.13.1

double **x**

X coordinate.

10.3.13.2

double **y**

Y coordinate.

10.3.14

void **DeleteAtIndex** (int index)

Arguments

10.3.14.1int **index** 165

Delete element by index.

10.3.14.1

int **index**

Element to delete.

10.3.15

void **DeleteNear** (double x, double y)

Arguments

Delete element near coordinate.

10.3.15.1

double **x**

X coordinate.

10.3.15.2

double **y**

Y coordinate.

10.3.16

int **ElementCount** () const

Return element count.

10.3.1

const int **initSize = 10**

Initial allocation size.

10.3.2

const int **growSize = 10**

Growth allocation.

10.3.3

Exit * **evect**

Element vector.

10.3.4

int **vSize**

Allocated size of vector.

10.3.5

int **vCount**

Number of used elements.

10.3.6

int **NearestIndex** (double x, double y) const

Arguments

Nearest index.

10.3.6.1

double **x**

X coordinate.

10.3.6.2

double **y**

Y coordinate.

10.4

class **Item**

Public Members

Private Members

Item in space.

10.4.1

enum **ItemType**

Members

Item types.

10.4.1.1 Character

Item is a Character.

10.4.1.2 Monster

Item is a Monster.

10.4.1.3 Treasure

Item is a Treasure.

10.4.1.4 TrickTrap

Item is a Trick or Trap.

10.4.1.5 Dressing

Item is some random other object.

10.4.1.6

Unspecified

Item has no specific type.

10.4.7

ItemType **Type** () const

Return type of item.

10.4.8

double **XCenter** () const

Return center X coordinate.

10.4.9

double **YCenter** () const

Return center Y coordinate.

10.4.10

const char * **Image** () const

Return image.

10.4.11

const char * **Filename** () const

Return file name.

10.4.12

Item (ItemType t = Unspecified, double x = 0.0, double y = 0.0, const char *im = "", const char *f = "")

Arguments

Constructor.

10.4.12.1

ItemType **t**

Item type.

10.4.12.2

double **x**

X coordinate.

10.4.12.3

double **y**

Y coordinate.

10.4.12.4

const char * **im**

Image.

10.4.12.5

const char * **f**

File name.

10.4.13

~Item ()

Destructor.

10.4.14

Item& **operator =** (const Item& source)

Assignment operator.

10.4.2

ItemType **type**

Type of item.

10.4.3

float **xCenter**

Center X coordinate.

10.4.4

float **yCenter**

Center Y coordinate.

10.4.5

const char * **image**

Image.

10.4.6

const char * **filename**

File name.

10.5

class **ItemVector**

Public Members

Private Members

Vector of Items.

10.5.7

ItemVector ()

Constructor.

10.5.8

~ItemVector ()

Destructor.

10.5.9

void **InsertItem** (const Item& source)

Arguments

10.5.9.1const Item&
source 176

Insertion function.

10.5.9.1

const Item& **source**

Item object to insert.

10.5.10

const Item* operator [] (int index) const

Arguments

10.5.10.1int **index** 177

Index operator.

10.5.10.1

int **index**

Index.

10.5.11

const Item* **Index** (int index) const

Arguments

10.5.11.1int **index** 177

Index function.

10.5.11.1

int **index**

Index.

10.5.12

const Item* **operator ()** (double x, double y) const

Arguments

10.5.12.1double **x** 178

10.5.12.2double **y** 178

Select nearest to (x,y) operator.

10.5.12.1

double **x**

X coordinate.

10.5.12.2

double **y**

Y coordinate.

10.5.13

const Item* **Nearest** (double x, double y) const

Arguments

Select nearest to (x,y) function.

10.5.13.1

double **x**

X coordinate.

10.5.13.2

double **y**

Y coordinate.

10.5.14

void **DeleteAtIndex** (int index)

Arguments

Delete element by index.

10.5.14.1

int **index**

Element to delete.

10.5.15

void **DeleteNear** (double x, double y)

Arguments

Delete element near coordinate.

10.5.15.1

double **x**

X coordinate.

10.5.15.2

double **y**

Y coordinate.

10.5.16

int **ElementCount** () const

Return element count.

10.5.1

const int **initSize = 10**

Initial allocation size.

10.5.2

const int **growSize = 10**

Growth allocation.

10.5.3

Item * **ivect**

Element vector.

10.5.4

int **vSize**

Allocated size of vector.

10.5.5

int **vCount**

Number of used elements.

10.5.6

int **NearestIndex** (double x, double y) const

Arguments

Nearest index.

10.5.6.1

double **x**

X coordinate.

10.5.6.2

double **y**

Y coordinate.

10.6

class **Space**

Public Members

Private Members

Basic Space class.

10.6.1

enum **SpaceShape**

Members

Space shape.

10.6.1.1

Square

Square.

10.6.1.2

Hexagon

Hexagon.

10.6.1.3

Undefined

Undefined shape.

10.6.13

void **UpdateFromRecord** (const Record &rec)

Update Space from Record.

10.6.14

Space (SpaceShape s = Undefined, double x = 0.0, double y = 0.0, const char *n = "", const char *d = "", const char *bg = "white")

Arguments

10.6.14.1SpaceShape	**s**	.....	186
10.6.14.2double	**x**	.....	186
10.6.14.3double	**y**	.....	186

Constructor.

10.6.14.1

SpaceShape **s**

Shape.

10.6.14.2

double **x**

Center point X coordinate.

10.6.14.3

double **y**

Center point y coordinate.

10.6.14.4

const char * **n**

Name of the space.

10.6.14.5

const char * **d**

Description of the space.

10.6.14.6

const char * **bg**

Background color of the space.

10.6.15

Space (const Space *that)

Copy constructor (from pointer).

10.6.16

Space (const Space &that)

Copy constructor (from reference).

10.6.17

~Space ()

Destructor.

10.6.18

operator const **Record** () const

Type conversion: convert to a Record.

10.6.19

SpaceShape **Shape** () const

Return this space's shape.

10.6.20

SpaceShape **SetShape** (SpaceShape newS)

Set the shape.

10.6.21

double **CenterX** () const

Return this space's center point X coordinate.

10.6.22

double **SetCenterX** (double newX)

Set the space's center point X coordinate.

10.6.23

double **CenterY** () const

Return this space's center point Y coordinate.

10.6.24

double **SetCenterY** (double newY)

Set the space's center point Y coordinate.

10.6.25

const Exit* **NearestExit** (double x, double y) const

Arguments

Return the nearest exit.

10.6.25.1

double **x**

X coordinate.

10.6.25.2

double **y**

Y coordinate.

10.6.26

const Exit* **IndexedExit** (int index) const

Arguments

Return the ith exit.

10.6.26.1

int **index**

Index.

10.6.27

int **NumberOfExits** () const

Return the count of exits.

10.6.28

void **InsertExit** (const Exit& source)

Insert an exit.

10.6.29

void **DeleteExitNear** (double x, double y)

Arguments

Delete exit nearest to x,y.

10.6.29.1

double **x**

X coordinate.

10.6.29.2

double **y**

Y coordinate.

10.6.30

void **DeleteExitAtIndex** (int index)

Arguments

Delete exit by index.

10.6.30.1

int **index**

Element to delete.

10.6.31

const Item* **Nearestitem** (double x, double y) const

Arguments

Return the nearest item.

10.6.31.1

double **x**

X coordinate.

10.6.31.2

double **y**

Y coordinate.

10.6.32

const Item* **IndexedItem** (int index) const

Arguments

Return the ith item.

10.6.32.1

int **index**

Index.

10.6.33

int **NumberOfItems** () const

Return the count of items.

10.6.34

void **InsertItem** (const Item& source)

Insert an item.

10.6.35

void **DeleteItemNear** (double x, double y)

Arguments

Delete item nearest to x,y.

10.6.35.1

double **x**

X coordinate.

10.6.35.2

double **y**

Y coordinate.

10.6.36

void **DeleteItemAtIndex** (int index)

Arguments

Delete item by index.

10.6.36.1

int **index**

Element to delete.

10.6.37

const char * **Name** () const

Return the name of the space.

10.6.38

const char * **SetName** (const char *newN)

Set the name of the space.

10.6.39

const char * **Description** () const

Return the description of the space.

10.6.40

const char * **SetDescription** (const char *newD)

Set the description of the space.

10.6.41

const char * **BackgroundColor** () const

Return background color of the space.

10.6.42

const char * **SetBackgroundColor** (const char *newBG)

Set the background color of the space.

10.6.2

SpaceShape **shape**

Space shape.

10.6.3

double **centerX**

Center point X coordinate.

10.6.4

double **centerY**

Center point Y coordinate.

10.6.5

ExitVector **exitList**

List of exits.

10.6.6

ItemVector **itemList**

List of items.

10.6.7

const char * **name**

Name of the space.

10.6.8

const char * **descr**

Description of the space.

10.6.9

const char * **bgcolor**

Background color of the space.

10.6.10

Record **rawData**

Internal storage of the space.

10.6.11

void **RecordToSpace** ()

Record to Space.

10.6.12

void **UpdateRecord** ()

Update Record.

11

References

[1] Gary Gygax. *Monster Manual*. TSR Games, Lake Geneva, WI, 1977, 1978.

[2] Gary Gygax. *Players Handbook*. TSR Games, Lake Geneva, WI, 1978.

[3] Gary Gygax. *Dungeon Masters Guide*. TSR Games, Lake Geneva, WI, 1979.

Class Graph