Dokumentasjon

Dungeon Warrior

av Sondre Husevold (s198755) og Magnus Tønsager (s198761)

Abstract_monster_factory

#include “abstract_monster_factory.hpp”

Beskrivelse:

Base klasse for alle monster factories. Har const verdier for hvor mye monstere sine stats skal øke pr level.

Medlemmer-public:

const double STAMINA_PR_LEVEL: andel monstere sin hp skal øke med pr level.

const double POWER_PR_LEVEL: andel monstere sin power skal øke med pr level.

const double ARMOR_PR_LEVEL: andel monstere sin armor skal øke med pr level.

Funksjoner-public:

abstract_monster_factory& operator=(const abstract_monster_factory& amf): copy assign operator

abstract_monster_factory& operator=(abstract_monster_factory&& amf): move assign operator.

virtual monster* create_monster()=0: returnerer et monster. Må overrides.

Block

#include “block.hpp”

Arv:

public clickable.

Beskrivelse:

Definerer en block i spillet. Dette er et synlig element som f.eks. gulv og vegg.

Medlemmer-private:

int ACCESS_: int for hva som skjer når dette omårdet aksesseres.

Funksjoner-public:

block(char icon, std::string info, int x, int y, const int ACCESS): ACCESS beskriver hva som skal skje når dette området aksesseres. icon, info, x og y sendes videre til clickable.

action_call left_click() override: override left_click, returnerer NO_ACTION

Clickable

#include “clickable.hpp”

Arv:

public visible.

Beskrivelse:

Base klasse for elementer som kan klikkes på. Garanter en action_call left_click() metode. typedef int action_call.

Medlemmer-public:

action_call konstanter, kan returneres av left_click()

static const action_call ATTACK

static const action_call LOO

static const action_call NO_ACTION

Funksjoner-public:

clickable(char icon, std::string info, int x, int y): konstruktør.

sender icon, info, x og y videre til visible

virtual action_call left_click()=0: returnerer en action_call konstant for hva som skal skje når dette elementet trykkes på. må overrides.

creature

Arver: clickable og reportable

Protected medlemmer:

std::vector<item*> : inventory;

std::string : name;

creature : target_;

int : stamina;

int : power;

int : armor;

int : base_stamina;

int : base_power;

int : base_armor;

int : hp_;

bool : in_combat_;

bool : has_target_;

Public funksjoner:

creature(std::string name_, char icon, std::string information, int x, int y);

creature(const creature& c);

creature(creature&& c);

creature& operator=(const creature& c);

creature& operator=(creature&& c);

int take_damage(int d);

int deal_damage();

bool attack();

void target(creature* target);

void move(int x, int y);

virtual bool dead();

bool in_combat()

bool has_target()

bool range();

int access() override;

void set_combat(bool c);

void heal(int h)

virtual std::string to_string()

std::vector<item*> get_items();

bool add_item(item* i);

void set_stamina(int stam);

void set_power(int pow);

void set_armor(int armour);

int get_stamina();

int get_power();

int get_armor();

int get_base_power();

int get_base_stamina();

int get_base_armor();

int get_hp();

Detaljert beskrivelse:

Creature er ment som en abstrakt klasse som både monster og player bruker. Her finne man ting som attributter monsteret/playeren har, angrepsmetoder, hva slags utstyr de har med seg, hvem de har i siktet, beveglsesmetoder og annet som slike elementer trenger for å fungere med hverandre.

Medlemsbeskrivelse:

std::vector<item*> : inventory;

Dette er alle tingene som dette vesnet har. Dette klassiferes som spillerens utstyrliste dersom det er player, eller hva monsteret kan droppe når det dør.

std::string : name;

Hva vesnet heter, spiller får spillernavnet fra MainWindow, mens monsterene har dette satt i factory.

creature : target_;

Hva vesnet fokuserer på i combat. Dette er spesielt brukt for monsterne ettersom de må "fokusere" på spilleren for å angripe han.

int : stamina;

Hvor mye maximum HP vesnet får fra utstyret.

int : power;

Hvor mye angrepsstyrke vesnet får fra utstyret.

int : armor;

Hvor mye rustningstyrke vesnet har fra utstyret.

int : base_stamina;

Hvor mye max HP vesnet har orginalt.

int : base_power;

Hvor mye angrepsstyrke vesnet har orginalt.

int : base_armor;

Hvor mye rustningstyrke vesnet har orginalt.

int : hp_;

Hvor mye health vesnet har.

bool : in_combat_;

Driver vesnet å kjemper akkurat nå? Hvis så, returnerer true.

bool : has_target_;

Fokuserer vesnet på noe? Hvis så, returner true.

Funksjonsbeskrivelse:

creature(std::string name_, char icon, std::string information, int x, int y);

Vanlig konstruktør til creature, tar imot et navn, et ikon for grafikk, beskrivelse av hvordan vesnet er og en x og y posisjon hvor det starter. Den initialiserer også alle medlemmene til creature via en initializer list.

creature(const creature& c);

Copy constructor til creature. Tar imot et creature og kopierer det.

creature(creature&& c);

Move constructor til creature. Tar imot et creature og flytter det.

creature& operator=(const creature& c);

Copy-assign operator for å kopiere et creature med copy.

creature& operator=(creature&& c);

Move-assign operator for å flytte creature via move constructor.

int take_damage(int d);

Denne metoden gjør at vesnet tar skade. Den tar da imot et visst antall skade som parameter, og vil bruke vesnets rustningstyrke for å senke antall skade den tar. Desto sterkere rustning, desto mer blir skaden dempet.

int deal_damage();

Denne returner skaden dette vesnet gjør. Det er base_power og power som da er våpenstyrken vesnet har.

bool attack();

Angriper target som vesnet har akkurat nå, men kun dersom vesnet faktisk er fortsatt i livet. Returnerer true dersom den klarte å angripe, og false dersom den ikke klarte det av grunner som f.eks at target allerede er død eller at target er for langt unna.

void target(creature* target);

Setter det creaturet som kommer med i parameter som primærtarget til dette vesnet.

void move(int x, int y);

Flytter vesnet med så mye i den retningen du setter. Dette fungerer som et koordinatsystem. F.eks move(1,0) vil få vesnet til å flytte seg til høyre, mens move(0,-1) vil flytte vesnet nedover.

virtual bool dead();

En virtuell metode som sjekker om vesnet er dødt. Denne er virtual fordi det er forskjell på når spilleren dør og monsteret dør da spilleren ikke skal droppe items f.eks.

bool in_combat()

Returnerer true dersom vesnet kjemper mot noe akkurat nå.

bool has_target()

Fokuserer vesnet på noe nå eller går det rundt? Hvis det fokuserer på noen så returnerer det true.

bool range();

Er target ved siden av dette vesnet? Hvis ikke returnerer det false og dette vesnet kan ikke angripe.

int access() override;

Hva skjer når du går på dette vesnet? Int har forskjellige konstanter i visible. Vanligvis er det ikke mulig å gå over vesnet med mindre det er dødt.

void set_combat(bool c);

Setter om vesnet kjemper mot noe eller ikke.

void heal(int h)

Gir vesnet mer liv. Brukt bl.a for player når han går rundt uten å være i combat.

virtual std::string to_string()

En virtual metode som gir en beskrivelse av vesnet. Dette er virtual for å gi forskjellige beskrivelser av spiller og monsterne.

std::vector<item*> get_items();

Returnerer inventory til dette vesnet. Brukt for inventory widgeten for spiller, og brukt for looting av monsternes utstyr.

bool add_item(item* i);

Legger til utstyr til vesnet. Brukt for player for å legge inn ting hos player når han looter utstyr, eller for monsterne når de initialiseres.

void set_stamina(int stam);

Setter en ny stamina verdi. Brukt spesielt hos player når han tar på seg rustning eller våpen med stamina på.

void set_power(int pow);

Setter en ny power verdi. Brukt spesielt hos player når han tar på seg ett våpen.

void set_armor(int armour);

Setter en ny armor verdi. Brukt spesielt hos player når han tar på seg rustning.

int get_stamina();

Get metode for stamina.

int get_power();

Get metode for power.

int get_armor();

Get metode for armor.

int get_base_power();

Get metode for orginal power.

int get_base_stamina();

Get metode for orginal stamina.

int get_base_armor();

Get metode for orginal armor.

int get_hp();

Get metode for nåværende health.

game_button

Arver: QPushButton

Public medlemmer:

game : gameplay;

Private medlemmer:

clickable : click_item;

Public funksjoner:

game_button(const QString & text, game* gameplay_, QWidget * parent = 0);

clickable: get_element();

void : set_element(clickable* c);

void : redraw();

Signals:

void : right_clicked();

Private slots:

void : mousePressEvent(QMouseEvent *e);

Public slots:

void : left_click_tile();

void : right_click_tile();

Detaljert beskrivelse:

Game_button er en knapp som har fått ytterligere datafelt enn vanlige QPushButton for å håndtere hva knappen skal "vise" og gjøre når den venstreklikkes eller høyreklikkes. Hver av disse er 128x128 med et ikon som tar opp hele knappen og er satt i et tabellformat i gui_handler. Det er 10x6 av disse knappene i Game widgeten til gui_handler for å vise spilleområdet.

Poenget med knappene er rett og slett å være dynamiske slik at knappene aldri slettes, men innholdet knappen gjør endres hver eneste gang noe oppdateres i spillet.

Medlemsbeskrivelse:

game : gameplay

Denne blir sendt fra gui_handler for å gjøre det mulig for knappen å sende clickable objektet videre til Game som tar hånd om hva som skjer når tingen blir trykket på.

Clickable : click_item

En av de viktigste delene av game_button er click_item. Alle blokker, creatures og andre elementer som er på en etasje i labyrinten er et clickable objekt. Dette objektet inneholder ting som ikonet, venstreklikk og høyreklikk actions, beskrivelser og annet som game_button kan bruke til å gjøre seg selv om til en slik ting og vise det å brettet.

Funksjonsbeskrivelse:

game_button(const QString & text, game* gameplay_, QWidget * parent = 0);

game_button sin konstruktør tar imot tekst, et game objekt og en forelder. Teksten og forelder blir sendt videre for å tilfredstille QPushButton sin konstruktør. Game blir initialisert i initializer listen og plassert i game : gameplay feltet for bruk i left click.

Konstruktøren setter også knappens ikon til 128x128 som er hele knappens størrelse og dens tekst til ingenting slik at teksten ikke tar over for ikonet.

clickable : get_element();

Henter ut click_item som ligger i denne knappen for videre bruk.

void : set_element(clickable* c);

Setter en ny click_item for å gjøre knappen helt annerledes.

void : redraw();

Oppdaterer hele knappen med ny grafikk

Signal beskrivelse:

void : right_clicked();

Brukes av mousePressEvent(QMouseEvent *e) for å sende ut et signal om at spilleren har høyreklikket på knappen og at den skal gi et signal at høyreklikk er aktivert.

Private slots beskrivelse:

void : mousePressEvent(QMouseEvent *e);

Denne metoden bruker QMouseEvent *e for å sjekke om det er venstreklikk eller høyreklikk som skjedde når spilleren trykket på knappen. Dersom det var et høyreklikk sender den ut et right_clicked() signal. Mens hvis det var noe annet sender den released() som da er venstreklikk.

Public slots beskrivelse:

void : left_click_tile();

Denne håndterer hva som skjer når knappen venstreklikkes. Den sender click_item videre til game objektet sin left_click_action metode for så å fullføre tingen den skulle gjøre der.

void : right_click_tile();

Samme som left_click_tile() bare at denne sender click_item sin beskrivelse av elementet (toString) direkte til report_to_log slik at det blir logget og forklart for brukeren når han høyre klikker på noe i spillet.

Game

#include “game.hpp”

Arv:

public reportable

Beskrivelse:

Representerer et spill. Holder en spiller og alle leveler. Har metoder for å utføre spilleren sine handlinger.

Members-private:

player* player_: spilleren.

std::vector<level*> levels_: holder alle levelene, nåværende og tidligere.

level* current_level_: nåvær level.

unsigned int current_: nummer på nåværende level, indeks mot levels_.

Funksjons-private:

void init(): initialiserings metode. Kalles av konstruktører.

Funksjons-public:

game(std::string player_name): oppretter et spill der player heter player_name.

game(player* player): oppretter et spill med player som spiller.

game(game& g): copy konstruktør

game(game&& g) move konstruktør

game& operator=(const game& g): copy assign operator.

game& operator=(game&& g): move assign operator.

void turn(): en turnering i spillet, skal kalles ett hver handling spiller gjør, som bevegelse eller angrep. Kaller curren_level_.turn()

void nex_level(): setter current level til neste level. Ny level blir generert hvis den ikke finnes fra før.

void prew_level(): går tilbake til forrige level hvis mulig.

player* get_player(): returnerer spilleren.

void left_click_action(clickable* left_click_object): ufører en handling på left_click_object ut fra hva left_click_object.left_click() returnerer.

void move player(int x, int y): flytter player x i x retning og y i y retning hvis mulig.

clickable* get_element(int x, int y): returnerer current_level_.get_element(x, y).

block* get_block(int x, int y): returnerer current_level_.get_block(x, y).

monster* get_monster(int n): returnerer curren_level_.get_monster(n).

gui_handler

Arver: QObject

Public medlemmer:

static const int : MAX_VIEWABLE_TILES_X = 10

static const int : MAX_VIEWABLE_TILES_Y = 6

bool : disableControls = false

static game : gameplay

static map : currentMap

static game_button : buttons[10][6]

static QTextEdit : command_log

Private medlemmer:

static QGraphicsView : hpBar

static QGraphicsScene : hpLeft

QLabel : weaponLabel

QLabel : armorLabel

QLabel : inventoryLabel

QLabel : playerLabel

QLabel : hpLabel

QLabel : powerLabel

QLabel : armorlabelTwo

QFrame : line

QListView : listView

static QLabel : hpEditable

static QLabel : powerEditable

static QLabel : armorEditable

QLineEdit : weaponEquipBox

QLineEdit : armorEquipBox

static QListWidget : inventoryBox

static QWidget : centralWidget

QWidget : gameWidget

QWidget : inventoryUI

QWidget : mapUI

Public funksjoner:

gui_handler(std::string playername, QWidget *centralWidget_, QWidget* inventoryUI_, QWidget* mapUI_, QObject *parent)

static QIcon : convert_to_icon(char charactericon)

static void : update_log()

static void : report_to_log(std::string sendtolog)

static game_button : get_button(int x, int y)

static void : redraw()

static void : check_current_inventory()

static void : update_hp_bar()

static void : you_died()

void : update_inventory_status_screen()

void : initialize_inventory()

Public slots:

void move_up()

void move_down()

void move_right()

void move_left()

void player_inventory_usage()

void redraw_map(int i)

Detaljert beskrivelse:

gui_handler tar hånd om det meste når det gjelder user interface og initialisering av spillet. Her vil selve spillbrettet bli initialisert ved å opprette en to-dimensjonal array av knapper som tilsvarer koordinater. Den vil opprette et nytt "game" objekt med spillernavn som parameter.

gui_handler oppretter også en logg som forklarer spilleren hva ting er og hva som skjer. Den har ansvar for å tegne opp spillbrettet for spilleren hver gang noe skjer. Den initaliserer inventory og map, den henter ut resursser som grafikk, den sjekker spillerens utstyr etter nye ting og tar hånd om når brukeren dobbeltklikker på en ting i utstyrslista. Den tar også hånd om health bar systemet, og viser spilleren når han døde.

Store deler av denne er static, grunnen til det er fordi når man først begynner å gjøre noe static vil resten også bli static. Redraw() metoden må være static og den er koblet med mye annet som vil føre til at det også er static. Mye av gui_handler må også hentes fra andre klasser og det skal bare finnes en instans av gui_handler om gangen.

Medlemsbeskrivelse:

static const int : MAX_VIEWABLE_TILES_X = 10

Dette er hvor mange knapper (spillområdet) du kan se i X retning om gangen.

static const int : MAX_VIEWABLE_TILES_Y = 6

Dette er hvor mange knapper (spillområdet) du kan se i Y retning om gangen.

bool : disableControls

Når spiller er død eller han har skiftet til inventory eller kartet vil denne settes som true slik at spilleren ikke kan gå rundt i spillet mens han er på andre skjermer enn "Game".

static game : gameplay

"game" klassen som tar hånd av mye spillrelaterte algoritmer. Se game dokumentasjon for mer info.

static map : currentMap

"map" klassen som tar hånd av kartskriving og oppdatering. Se map dokumentasjon for mer info.

static game_button : buttons[10][6]

Dette er alle spillebrikkene som vises som en grid i game. Hver av disse er en knapp slik at spilleren kan venstreklikke for å aktivere, angripe, ta opp ting og lignende. Og høyreklikke de for å få en beskrivelse av hva tingen er for noe. Dette er en to-dimensjonal array for å holde på koordinater. Se game_button for mer informasjon om hvordan knappen fungerer.

static QTextEdit : command_log

Loggen som vises på bunnen av Game widgeten. Her vil alle ting som skjer bli logget og vist til spilleren slik at spilleren vet hva som forgår.

static QGraphicsView : hpBar

Et QGraphicsView objekt som er beholder for hpLeft scenen. Disse tegner opp den lille health baren som er over spiller.

static QGraphicsScene : hpLeft

Et QGraphicsScene objekt som tegner opp health baren inni QGraphicsView.

QLabel : weaponLabel

Viser “Weapon: “ i inventory widgeten.

QLabel : armorLabel

Viser “Armor : “ i inventory widgeten.

QLabel : inventoryLabel

Viser “Inventory: “ i inventory widgeten.

QLabel : playerLabel

Viser en tjukk underlined tekst med “**Player:** “ i inventory widgeten.

QLabel : hpLabel

Viser en tekst med “HP: “ i inventory widgeten.

QLabel : powerLabel

Viser en tekst med “Power: “ i inventory widgeten.

QLabel : armorlabelTwo

Viser en tekst med “Armor: “ i inventory widgeten.

QFrame : line

En bakgrunn for å få statusen til spilleren til å “poppe” mer ut i inventory.

QListView : listView

Modellen til inventoryBox.

static QLabel : hpEditable

En tekst som oppdaterer seg når spilleren går inn i inventory fanen og viser spillerens nåværende HP.

static QLabel : powerEditable

En tekst som oppdaterer seg når spilleren går inn i inventory fanen og viser spillerens nåværende power.

static QLabel : armorEditable

En tekst som oppdaterer seg når spilleren går inn i inventory fanen og viser spillerens nåværende armor.

QLineEdit : weaponEquipBox

Viser en boks med nåværende brukt våpen. Denne er read only.

QLineEdit : armorEquipBox

Viser en boks med nåværende brukt rustning. Denne er read only.

static QListWidget : inventoryBox

En liste over alle tingene spilleren har plukket opp fra monstre. Denne er connected med "doubleClicked" signalet til QListWidget og gå til gui_handler sin player_inventory_usage() slot.

static QWidget : centralWidget

Dette er "Game" widgeten som holder alt i "Game" fanen. Den blir opprettet sendt med fra MainWindow.

QWidget : inventoryUI

Dette er "Inventory" widgeten som holder alt i "Inventory" fanen. Den blir opprettet og sendt med fra MainWindow, men initialisert i gui_handler i "initialize_inventory()" metoden.

QWidget : mapUI

Dette er "Map" widgeten som holder alt i "Map" fanen. Den blir opprettet og sendt med fra MainWindow.

Funksjonsbeskrivelse:

gui_handler(std::string playername, QWidget *centralWidget_, QWidget* inventoryUI_, QWidget* mapUI_, QObject *parent)

Konstruktøren til gui_handler som tar imot spillerens navn, "Game" widget fanen, Inventory widget fanen, Map widget fanen og eventuelt foreldren til gui_handler (MainWindow).

Denne initialiserer inventory UIen ved å bruke initialize_inventory() metoden. Oppretter et nytt "game" objekt, oppretter alle "tiles" knappene som spilleren ser som spillbrettet, den oppretter kommandologgen, health baren og dens scene, kartet og til slutt oppdaterer all grafikk som dermed skrives til knappene.

static QIcon : convert_to_icon(char charactericon)

Denne klassen konverterer en bokstav til et bilde av en ting som en vegg, bakken, figurene og lignende. Den returnerer da et QIcon objekt slik at knappene kan sette den som sitt ikon.

static void : update_log()

Henter en rapport fra game, sender den videre til report_to_log dersom den ikke er tom.

static void : report_to_log(**std::string sendtolog**)

Skriver ut til command_log, og passer på at det ikke er mer enn seks linjer i boksen om gangen, ellers vil den slette øverste linjen som en vanlig logg.

static game_button : get_button(**int x, int y**)

Henter ut en knapp på posisjon x og y.

static void : redraw()

Denne metoden kalles hver eneste gang noe skjer i spillet og vil oppdatere alle knappene med ny grafikk. Den setter da knappens "block" som det samme som er i level på den posisjonen spilleren er. Den vil også oppdatere loggen.

static void : check_current_inventory()

Sjekker inventory om det har kommet noe nytt der. Dersom det har det vil den oppdatere inventoryBox med den nye tingen ved å opprette et inventoryUIitem og legge det inn som en "widget".

static void : update_hp_bar()

Oppdaterer HP baren hver eneste runde. Den vil slette gamle scenen og opprette en ny scene med en rød rektangel inni med riktig størrelse på rektangelet for å vise spilleren hvor mye liv han har før han dør.

static void : you_died()

Kalles i game når spilleren dør. Det vil dukke opp et bilde som forklarer hva som har skjedd og to knapper vil opprettes som er connected til centralWidget sin foreldres foreldres foreldres foreldre som er da MainWindow slik at den kan bruke close() slotten og initialize_new_game() slotten. Grunnen til at den gjør dette vanskelig er for å unngå include loop.

void : update_inventory_status_screen()

Kalles når spilleren bytter til inventory fanen og vil oppdatere hva som står i statusen til player som hpEditable, powerEditable og armorEditable.

void : initialize_inventory()

Initialiserer hele inventory UIen. Den vil opprette inventoryBox, alle labelene, equipment boxene, sette teksten og lignende.

Public slots:

void move_up()

En slot koblet til MainWindow sitt keyboard_up QAction signal. Den vil da sende en instruks videre til game som setter spillerens posisjon opp.

void move_down()

En slot koblet til MainWindow sitt keyboard_down QAction signal. Den vil da sende en instruks videre til game som setter spillerens posisjon ned.

void move_right()

En slot koblet til MainWindow sitt keyboard_right QAction signal. Den vil da sende en instruks videre til game som setter spillerens posisjon en til høyre.

void move_left()

En slot koblet til MainWindow sitt keyboard_left QAction signal. Den vil da sende en instruks videre til game som setter spillerens posisjon en til venstre.

void player_inventory_usage()

En slot koblet til Inventory sin "inventoryBox". Når den blir dobbeltklikket på vil denne kalles som vil sende videre at spilleren skal bruke den tingen som ble dobbeltklikket på ved å hente ut tingen fra et inventoryUIitem.

void redraw_map(**int i**)

Sier til map at den skal oppdatere seg selv når spilleren bytter fane til "map" eller status til inventory. Int i vil sette om spilleren gikk til inventory eller map og kalle riktig metode som skal brukes og også fjerne kontroll muligheter til spillet.

inventoryUIitem

Arver: QWidget

Private medlemmer:

item : heldItem

Public funksjoner:

inventoryUIitem(item* i)

item : getItem()

void : setItem(item* i)

Detaljert beskrivelse:

På grunn av QListWidget sin implementasjon av ting i lista så må alt være QListWidgetItem noe som ikke er lett å få fiksa. Disse to skulle dermed vært et QListView og en QItemDelegate for å lage egendefinerte lister med egne objekter inni.

For å komme unna å lese all den avanserte dokumentasjonen til QListView og QItemDelegate ble det laget en omvei ved å misbruke QListWidget sin setItemWidget metode for å sette en widget inni itemet det er snakk om og ha denne som en slik widget.

Dermed var det mulig å sette inn items direkte inn i inventory sin inventoryBox, slik at tooltips ble vist riktig og man kunne aktivere de uten å måtte legge items i en egen vektor i gui_handler og iterere igjennom slike omveier.

Medlemsbeskrivelse:

item : heldItem

Dette er en peker til tingen som er i lista. Når spilleren dobbeltklikker på inventoryUIitemet i lista er det dermed lett å hente ut tingen fra spillerens inventory.

Funksjonsbeskrivelse:

inventoryUIitem(**item* i**)

Konstruktøren til inventoryUIitem. Den tar imot et item alltid for å sette som den tingen som blir holdt av dette listeobjektet.

Den setter også et tooltip for tingen slik at det er mulig å se hva utstyret gir spilleren når han setter det på.

item : getItem()

Henter ut utstyret som listeobjektet holder på som deretter kan equippes via player.

void : setItem(**item* i**)

Ubrukt metode, men er der i det tilfellet at et item må bli satt på nytt etter at det har blitt initialisert.

Item_database

#include “item_database.hpp”

Beskrivelse:

En database med alle itemer i spillet.

Medlemmer-private:

std::vector<item*> items_: en liste over alle items

Medlemmer-public:

standard verdier for ulike items. Brukes som en unik id mot hvert item og som søke verdi i get_item(int):

static const int COMMON_WEP

static const int COMMON_ARMOR

static const int LEGENDARY_WEP

static const int BLADE_BANISHER

static const int DEMON_WARGLAIVE

static const int DEMON_ARMOR

static const int PITLORD_ARMOR

static const int KINGS_BLADE

static const int TROLL_STAFF

static const int TROLL_ARMOR

static const int WIZZARD_ARMOR

static const int BLACK_DRAGON

static const int DRAGON_SLAYER

static const int DRAGON_ARMOR

static const int WYRM_ARMOR

static const int WARHAMMER

static const int ORC_BLADE

static const int ORC_ARMOR

static const int WAR_ARMOR

static const int ARMAGEDDON

static const int THUNDERFURY

static const int FROSTMOURNE

static const int SULFURAS=16

static const int PLATE_ARMOR

static const int DRAGONSKIN_ARMOR

static const int DEMONIC_ARMOR

static const int DESTROYER_ARMOR

Funksjoner-public:

item_database(): konstruktør

item_database(const item_database& id): copy construktør.

item_database(item_database&& id): move konstruktør.

item_database& operator=(const item_database& id): copy assig operator. item_database& operator=(item_database&& id): move assign operator.

Item

#include “item.hpp”

Beskrivelse:

Representerer et item i spillet. Har datafelter for ulike stats og metoder for å hente ut disse.

Medlemmer-public:

standardverdier for ulike item typer:

typedef int itemtype

static const itemtype WEAPON

static const itemtype ARMOR

static const itemtype CONSUMABLE

et unikt navn for itemet:

const int ITEM_NAME

Medlemmer-private:

std::string name: navnet på item.

itemtype type: item type, våpen, armor, osv.

bool dropped: true hvis itemet skal droppe.

int drop_rate_: sansynligheten i prosent for at itemet dropper.

int stamina: stamina verdi itemet gir.

int power: power verdi itemet gir.

int armor: armor verdi itemet gir.

Funksjoner-public:

item(const int IN, std::string name:, itemtype type, int drop_rate, int stam, int pow, int arm): konstruktør setter ITEM_NAME, name, type, drop_rate, stamina, power og armor ut fra parameterene.

item& operator=(const item& i): move assign operator. Kreves p.g.a non static const medlem.

bool drop(): returnerer true hvis itemet skal droppe, false ellers.

int get_stamina(): returnerer verdien for stamina itemet skal gi.

int get_power(): returnerer verdien for power itemet skal gi.

int get_armor(): returnerer verdien for armor itemet skal gi.

bool use_item(): bruker item hvis det er av typen consumable, isåfal returnerer true, ellers false.

itemtype get_type(): returnerer en itemtype konstant for type item.

std::string get_name(): returnerer itemets navn.

std::string to_string(): returnerer en string med navnet til item.

Level

#include “level.hpp”

Arv:

public report

Beskrivelse:

Representerer en level. Lager en labyrint av blocker og plasserer monstere i denne. Tar seg av kontrollen av monstere.

Medlemmer-private:

std::vector<block*> blocks_: holder alle blockene labyrinten består av.

block* null_block_: en standard “svart” blokk for udefinerte området, dvs utenfor labyrinten.

std::vector<monster*> monsters_: holder alle monstere i labyrinten.

abstract_monster_factory mf_: holder en monster factory, brukes til å lage monstere.

creature* player_: holder spiller. Brukes til å beregne posisjoner og monster handlinger.

Medlemmer-public:

static const int MAX_POSITION_X: maksimale posisjon i x-retning for blocker

static const int MAX_POSITION_Y: maksimale posisjon i y-retning for blocker

static const int ATTACK_RANGE: maksimale avstand det er mulig for creatures å angripe hverandre.

static const int COMBAT_RAMGE: maksimale avstand for hvor langt unna et monster må være player for å starte combat med player.

Funksjoner-private:

void create_road(std::vector<std::vecotor<int>> &map, int x, int y): lager en vei gjennom map med utgangspungt i (x, y).

void position_monsters(std::vector<std::vecotor<int>> &map, int x, int y):recursive metode for plassering av monstere.

void position_monsters(std::vector<std::vecotor<int>> &map): starter recursive plassering av monstere.

void create_blocks(std::vector<std::vecotor<int>> &map): lager blocker ut fra map.

void init(): initialiserings metode. Kalles av konstruktørene.

void follow_player(monster* monster): monster går en block mot player.

void walk_random(monster* monster): monster går i tilfeldig retning eller står stille.

Funksjoner-public:

level(creature* player, abstract_monster_factory* mf) :konstruktør, player er et creature som representerer spilleren. mf brukes til å lage monstere.

level(const level& l): copy konstruktør.

level(level&& l): move konstruktør.

level& operator=(const level& l): copy assign operator.

level& operator(level&& l): move assign operator.

void turn(): en "turnering" i spillet. Utfører monstere sine handlinger.

void visible_blocks(int x, int y, const int RANGE): gjør alle blocker i RANGE avstand fra (x, y) synlige.

void add_monster(int x, int y): bruker mf_ til å plassere et monster på (x, y) hvis denne posisjonen er gyldig.

bool valid_block_position(int x, int y): tester om (x, y) er en gyldig posisjon for en block, dvs om (x, y) er innenfor labyrintens rammer.

bool valid_monster_position(int x, int y): tester om (x, y) er en gyldig posisjon for et monster.

bool valid_position(int x, int y): tester om det er mulig å bevege seg til (x, y) for player.

static bool valid_range(creature &a, creature &b, const int RANGE): tester om avstanden mellom a og b er mindre eller lik RANGE.

clickable* get_element(int x, int y): returnerer det som er øverst på posisjon (x, y). creature>block.

block* get_block(int x, int y): returnerer blocken på posisjon (x, y), null_block hvis (x, y) er utenfor labyrintens grenser.

monster* get_monster(int n): returnerer monsteret på posisjon n i monsters_, nullptr hvis monsters_[n] er udefinert.

std::vector<std::vector<int>> create_bitmap(const int X, const int Y): git en X*Y array med int verdier satt til en stander verdi.

MainWindow

Arver: QMainWindow

Public medlemmer:

tabWidget : QTabWidget

keyboard_left : QAction

keyboard_right : QAction

keyboard_up : QAction

keyboard_down : QAction

keyboard_m : QAction

keyboard_i : QAction

Private medlemmer:

centralWidget : QWidget

Game : QWidget

Inventory : QWidget

Map : QWidget

title_screen : QWidget

playerName : QString

gui_handler_ : gui_handler

Private funksjoner:

void | initialize_title_screen()

Private slots:

void | change_to_map()

void | change_to_inventory()

void | initialize_new_game()

void | showAbout()

Detaljert beskrivelse:

Denne klassen tar hånd om initialisering av mye av user interfacet og har ansvar for startskjermen med “New Game” og “Exit”, og har også som oppgave å initialisere gui_handler som igjen starter et nytt spill. Den tar også hånd om å sette på plass tastatur snarveier for opp, ned, høyre, venstre, I og M knappene som spillet bruker og hva som vil skje når brukeren trykker på disse.

Medlemsbeskrivelse:

tabWidget : QTabWidget

Dette er en widget som tar hånd om fanene øverst i spillet. Den brukes for de tre tabbene “Game”, “Inventory” og “Map” og hver av disse får en QWidget hver.

keyboard_[knapp] : QAction

Dette er en QAction som kobler seg til tastaturet. Hver av disse blir connected til gui_handler sin “move_[retning]” slot, mens keyboard_i og keyboard_m blir connected til MainWindow sin change_to_inventory() og change_to_map() slots.

centralWidget : QWidget

Dette er hovedwidgeten som er sentrum av MainWindow klassen i begynnelsen. Den brukes blant annet til tittel skjermen og også for å holde på tabWidget når spillet initialisereres etter brukeren trykker “New Game” knappen.

centralWidget sin parent er MainWindow.

Game : QWidget

Hoved container for alle ting som skal i spill fanen. Her blir det gjerne puttet alle buttons fra gui_handler og command_log som viser loggen. Denne blir puttet inn som en fane i tabWidget. Den blir sendt med som “centralWidget” i gui_handler.

Inventory : QWidget

Hoved container for alle ting som skal i inventory fanen. Her blir det gjerne puttet ting som InventoryBox fra gui_handler, forskjellige equipment boxes, og status labels. Denne blir puttet inn som fane i tabWidget. keyboard_i har en slot som skifter til denne fanen. Den blir sendt med som "Inventory" i gui_handler for initialisering.

Map : QWidget

Hoved container for alle ting som skal i map fanen. Her blir det gjerne puttet ting som QGraphicsVieweren til kartet og dens scene. Denne blir puttet inn som siste fane i tabWidget. Den blir sendt med til gui_handler som "Map" for initialisering.

title_screen : QWidget

Hoved container for alle ting som skal vises på startskjermen. Inni denne blir det puttet ting som QPushButtons for "New Game" og "Exit". Den har en bakgrunn som er satt via CSS. Hver av dennes buttons er flate for å ta bort rammen dens og har fått satt på CSS per button for å gjøre den om til et bilde og skifte når musa holdes over.

To ting blir connected i title_screen, "New Game" button som connectes til initialize_new_game() slotten og "Exit" som er connected til "close()" slotten.

playerName : QString

Dette er spillernavnet som brukeren setter inn første gang de spiller spillet. Brukeren vil ikke bli spurt om å sette inn denne ytterligere uten å avslutte spillet helt og starte det på nytt. Denne blir sendt med videre til gui_handler.

gui_handler_ : gui_handler

Klassen som tar hånd av mye UI relaterte ting, den initialiserer også flere av fanene som MainWindow har startet. Se gui_handler klassen for mer informasjon om den.

Funksjonsbeskrivelse:

void | initialize_title_screen()

Denne initialiserer tittel skjermen i starten. Den initialiserer QWidgeten title_screen som den putter resten inni. Den lager to nye buttons - new_game_button og exit_button som kobles til "initialize_new_game()" slotten og "close()" slotten til QMainWindow. Hver button har fått et style sheet som endrer grafikken til knappen og fjerner rammen rundt.

Slot beskrivelse:

void | change_to_map()

Denne slotten bytter mellom map og game fanene. Dersom noe annet enn map er valgt vil den bytte til map fanen, og hvis den er i map allerede vil den gå tilbake til game fanen.

void | change_to_inventory()

Samme som change_to_map bare med inventory istedet.

void | initialize_new_game()

Denne slotten initaliserer et nytt spill. Den er koblet til "new_game_button" på tittelskjermen. Den sjekker om spilleren har satt inn et navn allerede eller aldri har gjort det før. Den vil deretter sjekke om det er et spill som pågår allerede og destruere alt det gamle spillet hadde for så å initalisere de på nytt som et nytt spill.

Den initialiserer tabWidget, Game, Inventory, Map QWidgetene, gui_handler klassen som starter et nytt spill, kobler alle spill knappene som er i gui_handler og setter alle tastatur snarveier.

void | showAbout()

Denne slotten viser about boksen som viser simpel informasjon om oss som lagde spillet og gir credit til de som lagde grafikken, et tileset kalt Spacefox orginalt laget for strategispillet Dwarf Fortress. Spacefox er free to use så lenge man gir de credits og ikke tar penger for det man bruker det til noe dette prosjektet ikke gjør.

map

Arver: QObject

Public medlemmer:

game : currentLevel;

Private medlemmer:

QWidget : centralWidget;

QGraphicsScene : tiles;

QGraphicsView : grapViewer;

QScrollArea : scrollings;

Public funksjoner:

map(QWidget* centralWidget_, game* currentLevel_, QObject * parent = 0);

void : set_up_map();

void : redraw();

QPixmap : convert_to_map_tile(char y);

Detaljert beskrivelse:

Map er kartet som blir tegnet opp for spilleren når han går inn i map fanen. Dette er et miniatyrkart som viser større områder av det spilleren har sett fra før. Hvis spilleren ikke har sett det før vil det ikke være mulig å se det på kartet ala. “fog of war”.

Map bruker egne ikoner og et QGraphicsView med en scene for å tegne opp kartet ettersom kartet ikke trenger å være trykkbare slik spillområdet i game er. Siden kartet da er på 101x101 tiles ville det tatt 2-3 sekunder på en kraftfull maskin å tegne opp dette hver gang spiller gikk til map dersom det var QLabels med ikoner eller QPushButtons. Dermed er det blitt brukt QGraphicsView for å optimalisere dette.

Medlemsbeskrivelse:

game : currentLevel;

Denne trengs for å kommunisere med game angående hva som er hvor og hvordan levelen er bygd opp. Uten denne kan ikke kartet bli tegnet opp ettersom den ikke vet hvordan etasjen er bygd.

QWidget : centralWidget;

Dette er den sentrale beholderen til map hvor map tegnes, som da vil være QWidgeten "Map" som er map fanen. Her vil alt som tegnes, bli tegnet.

QGraphicsScene : tiles;

Scenen til grapViewer, I denne tegnes alle bildene opp i rektangler ved bruk av QPixmaps. Denne tar dermed hånd om all tegning av kartet.

QGraphicsView : grapViewer;

Beholderen til GraphicsScenen "tiles". Denne presenterer scenen i central widgeten. Den fikser også scrollbars.

QScrollArea : scrollings;

Denne tar hånd om scrollbars og legges til i grapViewer. Viktig for andre OS som ikke lett kan scrolle til venstre og høyre ved bruk av en trackpad eller lignende touch overflate.

Funksjonsbeskrivelse:

map(QWidget* centralWidget_, game* currentLevel_, QObject * parent = 0);

Konstruktøren til map tar imot "Map" QWidgeten, et game for å lese av nåværende etasje i labyrinten og eventuelt et parent objekt for QObject sin konstruktør.

Den oppretter QScrollArea for scrolling, GraphicsVieweren, og den tegner bakgrunnen sort for ting som ikke har blitt "sett" enda av spilleren.

void : redraw();

Redraw tegner hele kartet om igjen ved å først slette den gamle tegnede scenen, også tegner opp 101x101 små 32x32 map tiles. Den setter også scrollbaren for den nye scenen helt av og scroller automatisk til det punktet spilleren er når den aktiveres. Denne metoden kalles når spilleren endrer fane til "Map".

QPixmap : convert_to_map_tile(char y);

Map har mindre tiles enn det game_button har for å gjøre det mer oversiktlig og brukbart som et kart. Disse er 32x32 imotsetning til game_button sin 128x128. Dermed må den konvertere chars på samme måte som gui_handler gjør, bare til map størrelse ikoner istedet. Denne returnerer QPixmaps da det er dette QGraphicsScene bruker for å vise bilder.

<Spesiell>_factory (monster_factorier)

#include "<spesiell>_factory.hpp"

Arv:

public abstract_monster_factory

Beskrivelse:

En factory for hvert monster. Spillet har følgende spesielle monster factorier:

- deathwing_factory
- monoroth_factory
- warlord_factory
- warrior_factory
- warscout_factory
- wrathguard_factory
- wyrmguard_factory

Hver returnerer et monster der monster navnet er det samme som <spessiel> i factory navnet. Hver factory har sine egne verdier for stats den tildeler monsteret.

Medlemmer-public:

int difficulty_: vanskelighetsgraden til monsteret.

Funksjoner-public:

<spessiel>_factory(int difficulty): difficulty blir vanskelighetsgraden for monsterene.

monster* create_monster() override: returnerer et <spessielt> monster.

Monster_factory

#include “monster_factory.hpp”

Arv:

public abstract_monster_factory

Beskrivelse:

Bruker andre spesielle monster factorier til å lage en blanding av monstere.

Medlemmer-private:

int difficulty_: monsterenes vanskelighetsgrad.

bool boss_spawned_: settes til true nå create_monster har returnert et monster med høyre vanskelighetsgrad enn vanlig.

std::vector<abstract_monster_factory*> mobs_: en liste over mulige monstere.

std::vector<abstract_monster_factory*> bosses_: en liste over mulige eksta vanskelige monstere.

Funksjoner-public:

monster_factory(int difficulty): konstruktør tar vanskelighetsgrad som parameter.

monster_factory(const monster_factory& mf): copy konstruktør.

monster_factory(monster_factory&& mf): move konstruktør.

monster_factory& operator=(const monster_factory& mf): copy assign operator.

monster_factory& operator=(monster_factory&& mf) move assign operator.

monster* create_monster() override: returnerer et tilfeldig monster fra mobs_ eller bosses_.

monster

Arver: creature, clickable, reportable, visible, position

Private medlemmer:

bool : looted_;

Public funksjoner:

monster(std::string name, char icon, std::string information, int x, int y);

std::vector<item*> : get_loot();

action_call : left_click() override;

bool : dead() override;

Detaljert beskrivelse:

Monster er et creature, og arver dermed alt fra creature. Men det har noen spesielle venstreklikk actions og egen dead() metode for å gi ut loot til spilleren.

Medlemsbeskrivelse:

bool : looted_;

Denne vil returnere true dersom monsteret er allerede looted og vil da gjøre at spilleren ikke plukker noe opp fra monsteret. Dersom det ikke er det vil alt lootet bli pushet til player dersom det dropper via get_loot metoden.

Funksjonsbeskrivelse:

monster(std::string name, char icon, std::string information, int x, int y);

Konstruktøren til monster. Denne er identisk med creature og vil sende alle parametere videre til creature konstruktøren til dette monstret. Den vil også sette looted som false under initialisering.

std::vector<item*> : get_loot();

Henter loot fra creature dersom det venstreklikkes på monsteret når det er dødt. Dersom spilleren allerede har looted monsteret vil det returnere en tom vector. Hvis ikke vil det pushes til player sitt inventory dersom utstyret droppet.

action_call : left_click() override;

action_call er en int type som har blitt definert i clickable. Dersom spilleren venstreklikker på monsteret vil det skje forskjellige actions basert på nåværende status til monsteret om det er dødt eller ikke.

bool : dead() override;

Returnerer true dersom monsteret er dødt, og endrer grafikk ikonet til monsteret til en loot bag dersom det droppa noe, og skjelett dersom det ikke droppet noe.

player

Arver: creature, clickable, reportable, visible, position

Private medlemmer:

bool : equiped_sword_;

bool : equiped_armor_;

item : sword_;

item : armor_;

Public funksjoner:

player(std::string _name, char icon_, std::string information_, int x_, int y_);

player(const player& p);

player(player&& c);

player& operator=(const player& p);

player& operator=(player&& p);

std::string : to_string() override;

action_call : left_click() override;

void : equip_item(item *i);

Detaljert beskrivelse:

Player er et creature og arver alt fra creature. Det har noen spesielle ting med seg blandt annet equipment, hva som skjer når det venstreklikkes på player og player sin beskrivelse.

Medlemsbeskrivelse:

bool equiped_sword_;

Sjekk om spilleren har et sverd equipped.

bool equiped_armor_;

Sjekk om spilleren har armor equipped.

item* sword_;

Hva spilleren har equipped som våpen.

item* armor_;

Hva spilleren har equipped som armor.

Funksjonsbeskrivelse:

player(std::string _name, char icon_, std::string information_, int x_, int y_);

Konstruktør for player, initialiserer creature ved bruk av paramterne den får inn, setter armor og sword equip til false siden spilleren starter naken, den initaliserer også spilleren med en target på seg selv for å forhindre segfaults og setter stamina power og armor.

player(const player& p);

Copy constructor til player.

player(player&& c);

Move constructor til player.

player& operator=(const player& p);

Copy-assign constructor til player.

player& operator=(player&& p);

Move-assign constructor til player.

std::string to_string() override;

Gir en beskrivelse av spilleren. Dette er alltid “This is you!”. Forhindre at spilleren får samme beskrivelse som andre creatures.

action_call left_click() override;

Hva som skal skje når spilleren venstreklikker på seg selv. I dette tilfellet er det NO ACTION.

void equip_item(item *i);

Spilleren tar på seg et utstyr. Det sjekkes for item type og den vil prøve å sette power og armor lik det utstyret som blir satt på.

Position

#include “position.hpp”

Beskrivelse:

Base klasse for alt som skal ha en posisjon. Har (x, y) koordinater

Medlemmer-private:

int x_: x koordinatet.

int y_: y koordinatet.

Funksjoner-public:

void set_x(int): Setter x koordinatet.

void set_y(int): Setter y koordinatet.

inline int x(): Returnerer x koordinatet.

inline int y(): Returnerer y koordinatet.

Reportable

#include “reportable.hpp”

Beskrivelse:

Raport klasse gir mulighet for å lagre og hente ut informasjon om hendelser. Gir en string med en raport per linje.

Medlemmer-private:

std::stringstream report_: holder det som blir raportert.

Funksjoner-public:

void report(const std::string s): skriver s til rapporten.

std::string get_report(): returnerer raporten med et linjeskift for hver raport lagt til. Sletter deretter raportert data.

Visible

#include “visible.hpp”

Arv:

public position

Beskrivelse:

Base klasse for alt som er synlig.

Medlemmer-public:

standard verdier for ulike ikoner:

static const char NAKED_PLAYER

static const char ARMORED_PLAYER

static const char LOOTABLE

static const char GHOST

static const char PRIEST

static const char ELF

static const char STAIR_UP

static const char FLOOR

static const char STAIR_DOWN

static const char CHEST

static const char ARMORED_CAPE_PLAYER

static const char CREATURE_BARD

static const char CREATURE_BOSS

static const char CREATURE_GIRL

static const char CREATURE_KING

static const char CREATURE_NINJA

static const char HOUSE

static const char DEFAULT_MONSTER

static const char SKELETON

static const char TREE

static const char WALL

Standard verdier for ulike hendelser når område aksesseres:

static const int NO_ACTION

static const int NEXT_LEVEL

static const int PREV_LEVEL

Medlemmer-private:

cahr icon_: representerer det synlige elementet.

std::string info_: informasjon om elementet.

bool visible_: synlig eller ikke synlig element.

bool accessable_: er det mulig å bevege seg over elementet.

bool moveable_: kan elementet bevege seg.

Funksjoner-public:

visible(char icon, std::string info, int x, int y): konstruktør.
x og y sendes videre til baseklassen position.

virtual int access()=0: returnerer en int konstant for hendelse når dette elementet aksesseres. Må overrides.

char get_icon(): returnerer icon_.

void set_icon(char c): setter c som nytt icon.

void set_visible(bool v): setter synlighet.

void set_accessable(bool a): setter mulighet for aksess.

void set_moveavle(bool m): setter mulighet for beveglese av elementet.

void set_info(std::string newinfo): endrer info_ til newinfo.

bool is_visible(): returnerer elementets synlighet

bool is_moveable(): returner om elementet er bevegelig.

bool is_accessable(): returnerer om det er mulig å bevege seg over elementet.

virtual std::string to_string(): returnerer innholdet av info_ medlemmet, kan overrides for å legge til mer informasjon.