The Defense of Dien Bien Phu

Objektorienterad Design (OOD)

version 0.1

2010-11-11

Fredrik Danielsson (Y3b)
Martin Danelljan (Y3b)
Gustav Jagbrant (Y3b)
Linus Envall (Y3b)
Simon Eiderbrant (Y3b)

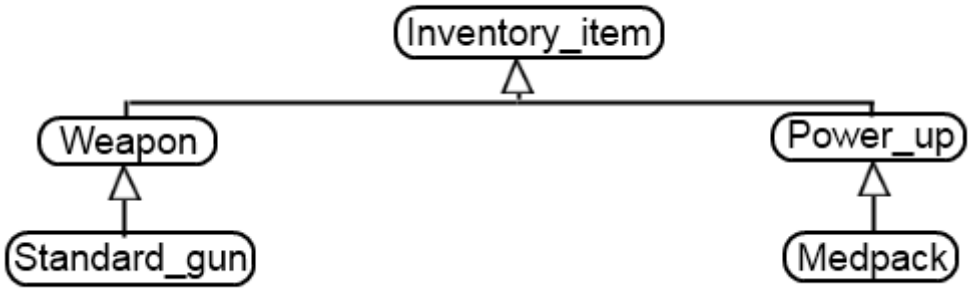
Sammanfattning

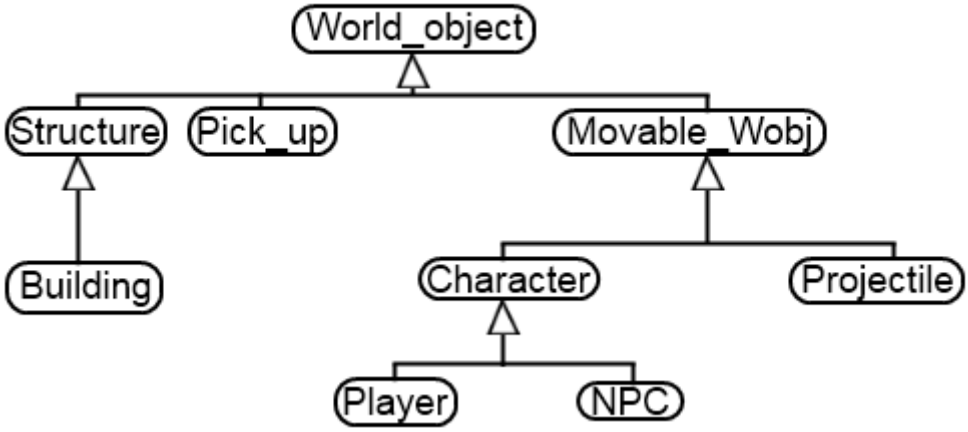
Detta projekt kommer att bestå av ett spel vid namnet The Defense of Dien Bien Phu, efter en by i Vietnam. Spelet går ut på att man ska försvara en barack mot anfallande bönder, soldater, m.m. Man kommer som spelare kunna ha vapen för att skjuta sina fiender och man ska kunna springa runt med piltangenterna. Spelet kommer att pågå till dess att fienden förstört en barack och spelaren sedan dött.

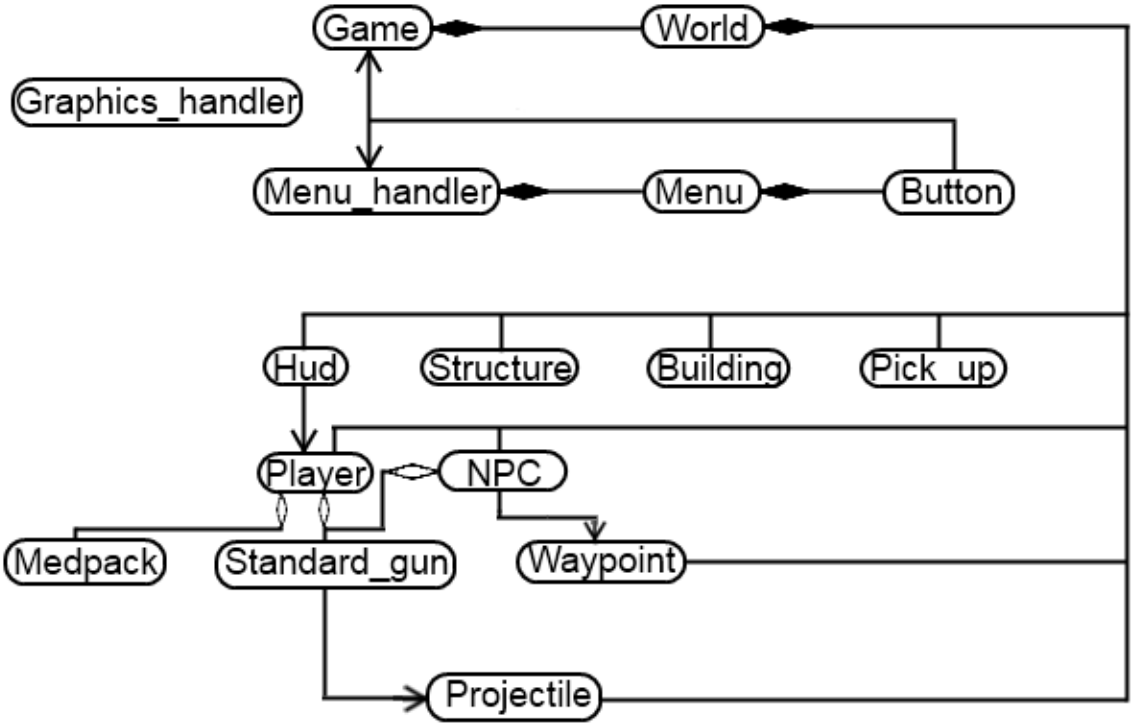
Detta dokument är resultatet av den objektorienterade designen för spelet. Dokumentet innehåller designbitar så som klassdiagram, klassbeskrivningar samt beskrivning av olika användningsfall.

Innehåll

1 Klassdiagram







2 Uppbyggnad

Vi tänker oss att ”överst” i vårt program ligger varsitt klass-objekt av Game och Menu_handler. Båda dessa initieras av huvudprogrammet. Menu_handler innehåller alla menyer och knappar och sköter denna del av spelet under tiden som game med sin viktiga datamedlem world sköter själva spelläget. All utritning på skärmen hanteras av klassen Graphics_handler som ligger som ett globalt objekt för att enkelt kunna kallas av alla klasser i programmet. Det finns också en global double variabel som fungerar som tidsfaktor för att enkelt kunna ändra spelhastigheten.

3 Klassbeskrivningar

3.1 Graphics_handler

Graphics_handler hanterar all utritning på skärmen. Den innehåller också texturerna till alla objekt i spelet. Texturerna ligger lagrade i ett map-objekt. Varje sorts objekt i spelet har en unik nyckel som kopplar den till rätt textur. Saker som inte bara är texturer som skall ritas ut har sina särskilda kommandon för utritning, t.ex om man vill rita ut en knapp.

3.1.1 Medlemsfunktioner

- Graphics_handler() – konstruktor.
- void init() – initierar grafikhanteraren och laddar in alla bilder.
- AUX_RGBImageRec* get_image(int image_key) – returnerar en pekare till ett bild_objekt givet en nyckel som är unik för varje gubbe.

- void draw(AUX_RGBImageRec*, int x_coord, int y_coord, int x_direction, int y_direction) – ritar ut ett objekt på givna koordinater med given riktning.

- void draw(string text,int size, int x_coord, int y_coord)

- void draw_button(string text, int x_coord, int y_coord, int height, int width)

- void draw_waypoint(Waypoint waypoint) – ritar ut en waypoint på kartan och alla waypoints som den är länkad till.

- void draw_map(int x_coord, int y_coord, int width, int height) – rutar ut den del av bakgrundsbilden som man vid en viss tidpunkt skall se på skärmen, x_coord och y_coord utgör nedre vänstra hörnet av bilden som ritas ut. Den uppdaterar också datamedlemmarna om skärmens koordinater.

3.1.2 Datamedlemmar

- map<int,AUX_RGBImageRec> images

- int left_x_

- int right_x _

- int bottom_y_

- int top_y_

3.2 Game

Game är det övergripande objektet då spelet körs. Med detta menas att det är i game som spelloopen finns och det är här som knapp- och mustryck från spelaren uppfattas och omvandlas till kommandon. Game har ett world-objekt som kommer att utföra alla kommandon som har med själva spelet att göra. Game kan också skicka kommandon till Menu_handler för att aktivera menyer.
.

3.2.1 Relaterade klasser

- Menu_handler

3.2.2 Medlemsfunktioner

- Game() – konstruktor.

- void set_menu_handler(menu_Handler* menu_handler) - sätter vilken meny-hanteraren spelloopen ska vara kopplad till.

- void start() – startar spelet och sätter igång spelloopen.

- void resume() – Sätter igång spelloopen igen efter att den varit pausad.
- void pause() – Stoppar spelloopen.
- bool started() – Returnerar true om spelet har startats.
- void end() – tar bort alla objekt i spelet, dvs. gör som att spelet aldrig startats.

3.2.3 Datamedlemmar

- World world

3.3 World

World innehåller alla objekt som finns i spelet och har kommandon för att uppdatera alla dessa samt utföra de kommandon som kommer från game.

3.3.1 Medlemsfunktioner

- World() – konstruktor.
- void init() – initierar alla objekt i världen.
- void reset() – tar bort alla objekt i världen.
- void stop_move(enum direction) – säger åt spelaren att sluta röra sig i en viss riktning.
- void start_move(enum direction) – säger åt spelaren att börja röra sig i en viss riktning.
- void shoot(int x_coord, int y_coord, bool held_down) – held_down är true om skottet kommer av att muspekaren är nertryckt och inte av att spelaren klickar.
- void spawn_wave(int wave_num) – spawnar våg nummer wave_num.
- void update_all() – uppdaterar alla objekt, dvs för spelet en loop framåt.
- void draw_all() – ritar bakgrundsbilden och alla objekt på skärmen.
- void set_previous_weapon()
- void set_next_weapon().
- void use_medpack()
- void set_weapon(int weapon_id)

3.3.2 Datamedlemmar

- Player player
- list<Projectile> projectiles_
- list<NPC> NPCs_
- list<Structure> structures_
- list<Building> buildings_
- vector<Waypoint> waypoints_
- int map_width
- int map_height
- int screen_width
- int screen_height
- int padding

3.4 Menu_handler

Menu_handler hanterar som namnet antyder, menyerna i spelet. Klassen har en egen loop och tolkar om key- & mouse-events och skickar vidare dessa till den aktuella menyn.
Menu_handler kan också hantera kommandon från knapparna för att byta meny.

3.4.1 Relaterade klasser

- Game

3.4.2 Medlemsfunktioner

- Menu_handler() – initierar meny-hanteraren tillsammans med alla menyer och knappar.
- void set_game(Game* game) – sätter vilket spelobjekt menyn ska vara kopplad till.
- void execute_command(command_number) – utför kommando.
- bool controle_command(command_number) – kontrollerar om ett visst kommando får utföras.
- void start_menu() – sätter igång meny_loopen.
- void exit_menu() – avslutar menyn återgår till game.

- void exit_program() – avslutar hela programmet.

3.4.3 Datamedlemmar

- vector<Menu> menues_
- Game* game_

3.5 Menu

Menu har button-objekt och ritar ut ett fönster utifrån dessa. Klassen hanterar kommandon från Menu_handler om var man klickat och vidarebefordrar detta till rätt button-objekt. Klassen hanterar också kommandon om att gå upp och ner bland knapparna och välja knapp med enter.

3.5.1 Medlemsfunktioner

- Menu(string name) – konstruktor.
- void add_button(Button new_button) – lägger till knapp till menyn.
- void press_button(int x_coord int y_coord) – aktiverar funktionen hos den knapp som ligger vid dessa koordinater.
- void draw() – ritar ut menyn.
- void move_up() – flyttar upp markören ett steg.
- void move_down() –flyttar ner markören ett steg.
- void press_button() – aktiverar den knapp som är markerad.

3.5.2 Datamedlemmar

- string name_
- vector<Button> buttons_
- int marked_button_

3.6 Button

Button initieras med en textsträng, en int paramater och en pekare till meny-hanteraren. Textsträngen utgör den text som skrivs ut på knappen. Int-parametern avgör vilken funktion i Menu_handler som skall kallas när man klickar på knappen. Pekaren gör så att knappen vet vilken meny-hanterare den ska kalla.
.

3.6.1 Relaterade klasser

- Game
- Menu_handler

3.6.2 Medlemsfunktioner

- Button(int command_number, string button_name, Menu_handler* menu_handler) – command_number avgör vilken funktion i Menu_handler som skall kallas när knappen klickas. Button_name är texten som skrivs ut på knappen.
- void do_command() – kallar funktionen i Menu_handler med sitt command_number.
- bool can_press() – returnerar true om knappens kommando kan utföras.
- string name() – returnerar namnet på knappen.

3.6.3 Datamedlemmar

- int command_number
- string name_
- Menu_handler* menu_handler_

3.7 HUD

HUD-klassen ritar ut information om spelaren på skärmen. Informationen som ritas ut är aktuellt vapen, hur mycket ammunition till aktuellt vapen som spelaren har, en cooldownmätare till vapnet, hur många medpacks han innehar och spelarens energimätare tillsammans med en exakt siffra.

3.7.1 Relaterade klasser

- Spelare

3.7.2 Medlemsfunktioner

- Hud(Player* player) – konstruktor.
- Draw() – ritar ut huden på skärmen.

3.7.3 Datamedlemmar

- Player* player_

3.8 World_object

World_object är den (abstrakta) klass som representerar alla typer av objekt som finns på banan.

3.8.1 Medlemsfunktioner

- World_object(int x, int y, AUX_RGBImageRec* texture) – konstruktor.
- ~World_object() – destruktor.
- void draw() – Ritar ut objektets textur på skärmen genom att anropa ett objekt av typ Graphics_handler.
- int get_x_pos()
- int get_y_pos

3.8.2 Datamedlemmar

- int xpos_
- int ypos_
- AUX_RGBImageRec* texture

3.9 Pick_up

Detta är den (abstrakta) klass som representerar ett objekt liggande på banan som går att plocka upp av spelaren. Den är en subklass till World_object.

3.9.1 Medlemsfunktioner

- Pick_up(int x, int y, AUX_RGBImageRec* texture, Inventory_item* item, int uses) – konstruktor.
- ~Pick_up() – destruktor.
- Inventory_item* get_item() – Returnerar en pekare till föremålet.
- int uses() – Returnerar antal ”uses”.

3.9.2 Datamedlemmar

- Inventory_item* item_
- int count_

3.10 Structure

Structure är den subklass till World_object som representerar konstruktioner på banan. Den kan antingen gå eller inte gå att skjuta genom, men aldrig att gå genom.

3.10.1 Medlemsfunktioner

- Structure(int x, int y, AUX_RGBImageRec* texture, int width, int height, bool penetrable = false) – konstruktor.
- ~Structure() – destruktor.
- bool penetrable() – Returnerar sant om objektet ska kunna skjutas genom av projektiler, annars falskt.
- int get_width()
- int get_height()

3.10.2 Datamedlemmar

- bool penetrable_
- int width_
- int height_

3.11 Building

Building är den subklass till Structure som representerar byggnader på banan. Dessa går att förstöra till skillnad från Structure-objekt.

3.11.1 Medlemsfunktioner

- Building(int x, int y, AUX_RGBImageRec* texture, AUX_RGBImageRec* blown_up_texture, int max_health, bool penetrable) – konstruktor.
- ~Building() – destruktor.
- void damage(int) – Skadar byggnaden med ett visst värde.
- bool destroyed() – Returnerar sant om objektet är förstört, annars falskt.
- bool passable_if_destroyed()– Returnerar om objektet ska kunna passeras om det är förstört.

3.11.2 Datamedlemmar

- Bool destroyed_

- bool passable_if_destroyed_
- int health
- AUX_RGBImageRec* destroyed_texture_

3.12 Movable_Wobj

Movable_Wobj är en abstrakt subklass till World_object. Den är basklass till alla objekt i världen som kan förflytta sig. Character och Projectile är de direkta subklasserna till Movable_Wobj.

3.12.1 Medlemsfunktioner

- Movable_Wobj(int x, int y, AUX_RGBImageRec* texture, int dir_x, int dir_y, int vel_x, int vel_y, radius) – konstruktor.
- ~Movable_Wobj() – destruktor.
- bool collides(World_object*) – Testar om objektet kolliderar med ett annat World_object.
- virtual bool update() – Uppdaterar objektets position, samt utför kollisionshantering.
- void set_vel_x(int)
- void set_vel_y(int)
- int get_vel_x()
- int get_vel_y()
- void set_dir_x(int)
- void set_dir_y(int)
- int get_dir_x()
- int get_dir_y()
- int get_radius()

3.12.2 Protected

- virtual void move() – // Ska den vara virtual? Kan tänkas att någon typ av objekt ska förflyttas annorlunda än andra. Förflyttar ett objekt med dess hastighet.

3.12.3 Datamedlemmar

- int radius_ – Radie för objektets utsträckning. Alla förflyttbara objekt kommer alltså att vara cirkulära.
- int vel_x_ – Hastighetsvektor x-led
- int vel_y_ – Hastighetsvektor y-led
- int dir_x_ – Riktningsvektor x-led
- int dir_y_ – Riktningsvektor y-led

3.13 Projectile

Projectile är en subklass till Movable_Wobj. Den ska representera avfyrade skott i världen.

3.13.1 Medlemsfunktioner

- Projectile(int x, int y, AUX_RGBImageRec* texture, int dmg, int lifelength) – konstruktor.
- ~Projectile() – destruktor
- int get_dmg() – Returnerar hur mycket skada projektilen gör.
- int get_lifelength() – Returnerar projektilens livslängs, dvs hur lång tid tills den försvinner från banan.
- int get_rem_life()

3.13.2 Datamedlemmar

- int lifelength_
- int damage_

3.14 Character

Character är en abstrakt subklass till Movable_Obj. Den är basklass till alla karaktärer (”gubbar”) i spelet, inklusive spelaren själv. Subklasserna till Character är Player och NPC. Character innehåller främst funktionalitet för inventoryn och för att lagra attribut.

3.14.1 Relaterade klasser

- Inventory_item och dess subklasser

3.14.2 Medlemsfunktioner

- Character(double x, double y, AUX_RGBImageRec* texture, double r, double acceleration, enum team, string name, int level, int standard_weapon_skill, int aim_skill, int endurance, int speed_rating) – konstruktor.

- ~Character() – destruktor.

- enum get_team() – Returnerar vilket lag karaktären är med i.

- string* get_name() – Returnerar karaktärens namn, till exempel om denna är en spelare eller en viss typ av fiende eller vän.

- bool dead() – Returnerar true om karaktären är död, annars false.

- double get_health() – Returnerar karaktärens aktuella hälsa (health).

- double get_max_health() – Beräknar och returnerar karaktärens maximala hälsa (health). Denna beror endast på endurance_.

- double get_max_velocity() – Beräknar och returnerar karaktärens maximala fart. Denna beror endast på speed_rating_.

- int get_attribute(enum attribute) – Returnerar karaktärens värde på attributet attribute.

- Weapon* get_current_weapon() – Returnerar en pekare till det vapnet som karaktären har framme vid tillfället.

- void give_weapon(Weapon weapon) – Ger vapnet weapon till karaktären.

- int get_current_ammo() – Returnerar hur mycket ammunition som finns kvar till det vapnet som karaktären har framme vid tillfället.

- void increase_ammo(int amount, enum ammo_type) – Ökar mängd ammunition av typen ammo_type med amount. Blir en minskning om amount är mindre än noll.

- void set_ammo(int amount, enum ammo_type) – Sätter mängd ammunition av typen ammo_type till amount.

- int get_power_ups(enum power_up_type) – Returnerar antal power ups av typen power_up_type som karaktären har kvar.

- void increase_power_ups(int amount, enum power_up_type) – Ökar antal power ups av typen power_up_type med amount. Blir en minskning om amount är mindre än noll.

- void set_power_ups(int amount, enum power_up_type) – Sätter antal power ups av typen power_up_type till amount.

3.14.3 Datamedlemmar

- enum team_
- string name_
- int level_
- bool dead_
- double health_
- double velocity_
- double acceleration_ – Ett negativt tal betyder ”oändlig” acceleration.
- int standard_weapon_skill_ – Påverkar skadan som görs med standard vapen (objekt av Standard_weapon).
- int aim_skill_ – Påverkar hur bra man siktar med vapnet.
- int endurance_ – Ökar den maximala hälsan.
- int armor_rating_ – Minskar skadan som karaktären tar av en projektil på ett sätt som inte är proportionellt med grundskadan som projektilen gör.
- int speed_rating_ – Ökar karaktärens maximala hastighet.
- vector<Weapon> weapon_list_
- vector<Power_up> power_up_list_
- vector<int> ammo_list_ – Vektor med mängd ammunition av de olika typerna. Ett negativt tal för en viss typ betyder oändligt med ammunition av denna typ.
- Weapon* current_weapon_
- int weapon_cooldown_

3.15 Player

Player är en subklass till Character. Den ska representera användarens karaktär i spelet. Player innehåller främst funktionalitet för att reagera på events, kollisionshantering och levling.

3.15.1 Medlemsfunktioner

- Player(double x, double y, AUX_RGBImageRec* texture, double r, double acceleration, int standard_weapon_skill, int aim_skill, int endurance, int speed_rating) – konstruktor.

- ~Player() – destruktor.

- void set_motion(enum motion_type) – Gör player-objektet medvetet om att användaren vill flytta spelaren på sättet motion_type. Sätten att förflytta spelaren är: gå framåt, gå bakåt, gå i sidled höger och gå i sidled vänster.

- void stop_motion(enum motion_type) – Gör player-objektet medvetet om att användaren inte längre vill flytta spelaren på sättet motion_type. Sätten att förflytta spelaren är: gå framåt, gå bakåt, gå i sidled höger och gå i sidled vänster.

- void update_aim_direction(double mouse_x , double mouse_y) – Sätter spelarens riktning (riktningen spelaren siktar).

- void shoot(double mouse_x , double mouse_y, bool held_down) – Hanterar avfyrning av spelarens vapen och uppdaterar samtidigt spelarens riktning (riktningen spelaren siktar). held_down är false om spelaren precis tryckte ned avfyrningsknappen, annars true.

- void set_next_weapon() – Byter till nästa vapen i ordningen bland de vapen som spelaren har.

- void set_previous_weapon() – Byter till föregående vapen i ordningen bland de vapen som spelaren har.

- void set_weapon(enum weapon) – Byter till vapnet med identifikationen weapon om denna finns hos spelaren.

- void update(list<Structure>* structure_list, list<Building>* building_list, list<NPC>* NPC_list) – Förflyttar spelaren. Först beräknas spelarens hastigheter utifrån vilka tangenter som är nedtryckta och acceleration. Sedan upptäcks och hanteras kollisioner med alla objekt som ges i listorna. Uppdaterar också vapnets cooldown.

- void handle_projectile_collision([container]<Projectile>* projectile_list) – Upptäcker och hanterar kollisioner med alla projectiles i projectile_list.

- void handle_pick_up_collision([container]<Pick_up>* pick_up_list) – Upptäcker och hanterar kollisioner med alla pick_ups som ges i pick_up_list.

- bool increase_experience(int exp) – Ökar spelarens experience med exp. Om spelaren går upp i level returneras true, annars false. Då spelaren går upp i level ökas leveln med ett och spelarens attribut ökas med lämpliga värden. När leveln ökas uppdateras också max_health_ och max_velocity_.

- void respawn(double x, double y, double dir_x, double dir_y) – Återupplivar spelaren och placerar ut honom i världen på angivna koordinater och med angiven riktning.

3.15.2 Datamedlemmar

- bool move_forward_
- bool move_backward_
- bool strafe_right_
- bool strafe_left_
- int experience_

3.16 NPC

NPC (Non Player Character) är en abstrakt subklass till Character. Den representerar alla datorstyrda karaktärer i spelet. NPC innehåller främst funktionalitet för kollisionshantering och attackering av fiender.

3.16.1 Relaterade klasser

- Waypoint
- Character
- Building

3.16.2 Medlemsfunktioner

- NPC(double x, double y, AUX_RGBImageRec* texture, double r, double acceleration, enum team, string name, int level, int standard_weapon_skill, int aim_skill, int endurance, int speed_rating, Waypoint* start_waypoint) – konstruktor.
- ~NPC() – destruktor.
- void update(list<Structure>* structure_list, list<Building>* building_list, list<NPC>* NPC_list, Player* player) – Förflyttar NPC:n. Först Beräknas NPC:ns hastigheter och riktning utifrån bland annat waypoints som ska följas och acceleration. Sedan upptäcks och hanteras kollisioner med alla objekt som ges i listorna. Uppdaterar också vapnets cooldown.
- int handle_projectile_collision([container]<Projectile>* projectile_list) – Upptäcker och hanterar kollisioner med alla projectiles i projectile_list. Om NPC:n dör hanteras detta och mängd experience spelaren ska få returneras. Annars returneras -1. Mängd experience som spelaren ska få beror på NPC:ns level.
- bool attacking() – Returnerar true om NPC:n attackerar en fiende, annars false.

- bool attack(list<Building>* building_list, list<NPC>* NPC_list, Player* player) – NPC:n försöker attackera någon fiende. Detta görs genom att NPC:n först försöker attackera fienden som är dess ”current target” (nuvarande fiende i sikte). Om inte detta går eller om ”current target” är en byggnad, försöker NPC:n attackera någon fiendekaraktär i enemy_character_list. Om heller detta går försöker NPC:n attackera fiendespelaren. Om heller detta går försöker NPC:n attackera någon fiendebyggnad i enemy_building_list om detta går.

3.16.3 Datamedlemmar

- Waypoint* current_waypoint_
- bool avoiding_obstacle_
- bool avoiding_right_
- Character* current_character_target_
- Building* current_building_target_

3.17 Waypoint

Waypoint ska representera stationer i den utstakade vägen som NPC:s ska gå mot. När en NPC kommer tillräckligt nära ett Waypoint-objekt ska den börja gå mot nästa. Nästa waypoint ges av Waypoint-objektet som NPC:n precis kom fram till.

3.17.1 Medlemsfunktioner

- double get_xpos() – Returnerar wapointens x-koordinat.
- double get_ypos() – Returnerar wapointens y-koordinat.
- double get_radius() – Returnerar radien inom vilken NPC:n betraktas ha kommit fram till denna waypoint och istället ska börja gå mot nästa.
- Waypoint* get_next_waypoint() – Returnerar en pekare till nästa waypoint som NPC:n ska gå mot. Om en tom pekare returneras betyder det att NPC:n har kommit till slutet av rutten och alltså ska stå still.

3.17.2 Datamedlemmar

- double xpos_
- double ypos_
- double radius_
- Waypoint* next_waypoint_

3.18 Inventory_item

En abstrakt klass som representerar ett objekt som spelaren har i sitt ”inventory”. Till exempel om spelaren plockar upp ett vapen eller ett ”medpack” från marken så skapas ett Inventory_item och läggs till i spelarens inventory-lista. Basklass till Weapon och Power_up.

3.18.1 Medlemsfunktioner

- virtual Inventory_item() = 0 – ”Pure virtual” konstruktor.
- virtual ~Inventory_item() = 0 – ”Pure virtual” destruktor.
- string getTexture() – Returnerar namnet på det texturobjekt i graphics-handler som tillhör detta inventory item

3.18.2 Datamedlemmar

- string texture_ – Namnet på det texture-objekt som tillhör aktuellt inventory_item. Ett texture-objekt är direkt mappat till denna sträng i graphics-handler.

3.19 Weapon

En abstrakt klass som representerar ett vapen som spelaren har. Subklass till Inventory_item och basklass till Standard_Gun och andra vapen som eventuellt kommer att läggas till.

3.19.1 Medlemsfunktioner

- virtual Weapon() = 0 – ”Pure virtual” konstruktor.
- virtual ~ Weapon () = 0 – ”Pure virtual” destruktor.
- Weapon(int dmg, int cool, bool auto, int life_time) – Sätter de olika datamedlemmarna. Anropas av subklasser.
- virtual vector<projectile> shoot() – Returnerar en vektor med projektilobjekt som läggs i en lista i game. Spel-loopen tar sedan hand om projektilernas förflyttningar och kollisionshantering.

3.19.2 Datamedlemmar

- int damage_ – Anger hur stor skada ett skott ifrån vapnet i fråga ger.
- int cooldown_ – Anger tiden mellan två skott.
- bool automatic – Anger om vapnet är automatiskt (true) eller semiautomatiskt (false)
- User_Enum user – Anger vem som använder vapnet. User_Enum är en enumerator som innehåller värdena Player, Helper och Enemy.
- Ammo_Enum ammo_type – Anger vilken typ av ammunition som används med vapnet.

- int projectile_lifetime – Anger hur långt ett skott ska kunna gå.

3.20 Standard_gun

Standard_gun representerar ett standardvapen som används av spelaren, medhjälpare och fiender. Med hjälp av olika värden på parametrar kan vapen med olika egenskaper skapas. Subklass till Weapon.

3.20.1 Medlemsfunktioner

- Standard_Gun(int dmg, int cool, bool auto, int lifetime, int no_bullets) – Sätter ammo_count och anropar Weapons konstruktor med parametrarna i parameterlistan och false för auto-parametern.

- vector<projectile> shoot() – Returnerar en vektor med projektilobjekt.

- void set_angle(int angle) – Sätter angle till ett heltal mellan 0 och 360.

3.20.2 Datamedlemmar

- int number_of_bullets_ – Anger antal skott som avfyras för varje musklick. Det kommer alltså vara 1 för en pistol och fler för exempelvis hagelgevär

- int angle_ – Anger vilken riktning de olika skotten kommer att spridas från skottriktningen.

3.21 Power_up

En abstrakt klass som representerar ett objekt i spelarens inventory som kan användas för att förändra, ev. förstärka, en egenskap hos spelaren. Subklass till Inventory_item och basklass till Medpack och andra Power_up variationer som eventuellt läggs till senare.

3.21.1 Medlemsfunktioner

- virtual Power_up() = 0 – ”Pure virtual” konstruktor.

- virtual ~ Power_up() = 0 – ”Pure virtual” destruktor.

3.21.2 Datamedlemmar

- Power_up_enum power_type_ – Anger vilken typ av power_up det är.

3.22 Medpack

Klassen representerar de första hjälpen-lådor som kan användas av spelaren. Den är subklass till Power_up.

3.22.1 Medlemsfunktioner

- Medpack(int am) – Initierar amount och anropar Power-ups konstruktor.
- void use() – Lägger till på spelarens hälsa och kollar så att hälsan inte går över 100%.

3.22.2 Datamedlemmar

- int amount_ – Anger hur mycket extra hälsa spelaren ska få av att använda medpacket.

4 Användningsfall

4.1 Spelet startas för första gången

Menu-handler säger till startmenyobjektet att rita upp sig själv. Vid klick på ”instructions” utförs knappens do_command vilket anropar ett kommando i menu_handler som i sin tur öppnar instruktionssidan/-menyn och stänger startmenyn. När spelaren klickar på ”tillbaka” utförs funktionen i ”tillbaka”-knappen. Denna anropar ett kommando i menu_handler som ändrar tillbaka till startmenyn.

När spelaren klickar på ”start” kör knappen funktionen game.start. Funktionen säger då åt World-objektet att initiera världen och går in i en oändlig spelloop.

För varje varv i loopen anropar game ett kommando i world som anropar en update-funktion hos de objekt som är rörliga i världen. Innan dess har game hanterat events (knapptryckningar, mustryckningar) och anropat motsvarande funktioner i world och menu_handler och tagit hand om kollisionshantering för alla objekt.

4.2 Spelaren trycker på ”escape”-knappen under spelets gång

Events-hanterarfunktion i början av spelloopen upptäcker escape-tryckningen och anropar menyhanterarloopen. Vid ”resume”-klick anropar knappen spelloopen igen.

4.3 Spelaren klickar på musknapp eller trycker på piltangent

Players” medlemsfunktioner set_motion, stop_motion och shoot anropas.

4.4 Spelaren dör

- Om baracken finns kvar: ”Player ” återinitieras genom att Player.respawn anropas.
- Om baracken inte finns kvar: Game återgår till startmenyn.