PEMROGRAMAN 4

ANALISA CLASS DALAM GAME



Disusun oleh :

Achmad Robith Fuadi 4210161027

**PROGRAM STUDI TEKNOLOGI GAME**

**DEPARTEMEN TEKNOLOGI MULTIMEDIA KREATIF**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

**SURABAYA**

**2018**

* Game Roguish-master (open source)

Class

* Form

var formCounter = 0;

/\*\*

\* the form class adalah yang bertanggung jawab merender input html ke dom dan memberikan respon terhadap input ini berupa interface informasi.

\* @param {number} x coorddinate

\* @param {number} y coordinate

\* @param {Function} list of fields

\* @param {Object} @optional options object

\*/

var Form = function(x, y, fields, options) {

this.formName = "default-form-" + formCounter + "-name-";

formCounter++;

// if an object is passed in for x, then we know it is actually the fields and shift the arguments over

if (typeof(x) === "object") {

options = y;

fields = x;

x = 0;

y = 0;

}

this.fields = fields;

this.header = options ? options.header || null : null;

this.message = options ? options.message || null : null;

this.cssClass = options ? options.cssClass || null : null;

// inherited constructor

Widget.apply(this, [{x: x, y: y, cssClass: this.cssClass}]);

/\*\*

\* [getValue gets the value of an internal form field]

\* @param {value}

\* @return {string}

\* @public

\*/

this.getValue = function(value) {

return this.el.find("#" + value).val();

};

/\*\*

\* [values returns the values of all fields within a form]

\* @return {Object}

\* @public

\*/

this.values = function() {

obj = {};

this.fields.forEach(function(field) {

obj[field.text] = this.getValue(field.text);

}, this);

return obj;

};

/\*\*

\* [render renders the form to the screen]

\* @public

\*/

this.render = function() { fungsi merender form ke layar

this.el.html(template.call(this));

if(this.message){

this.el.prepend("<p>"+this.message+"</p>");

}

if(this.header){

this.el.prepend("<h2>"+this.header+"</h2>");

}

this.container.append(this.el);

attachCallbacks.call(this);

};

// private

/\*\*

\* [getId gets the idea of a form field by its number]

\* @param {number|string} index

\* @return {string}

\* @private

\*/

function getId(index) { fungsi menemukan suatu id data dalam index

if (typeof(index) === "number") {

return this.formName + index.toString();

} else {

return index;

}

}

/\*\*

\* [template returns the proper template for the desired form.]

\* @return {string}

\* @private

\*/

function template() { mereturn template lalu memanggil yang sesuai dengan apa yang diinput

return inputFields.call(this);

}

/\*\*

\* Displays raw text on the form field

\* @param {string} textToRender The string of text to render

\* @return {string} HTML code to be injected to display text

\* @private

\*/

function basicText(textToRender) { fungsi merender text

return "<p class='basic-text'>" + textToRender + "</p>";

}

/\*\*

\* [textField diplays a text field]

\* @param {string} name

\* @param {number} index

\* @return {string}

\* @private

\*/

function textField(name, index) { fungsi menampil textField

return "<input class='input-default' placeholder='" + name + "' id='" + getId.call(this, index) + "' autofocus ></input>";

}

/\*\*

\* [buttonField displays a button field]

\* @param {string} name

\* @param {number} index

\* @return {string}

\* @private

\*/

function buttonField(name, index) { fungsi sebagai button nama dalam field

return "<button class='button-default' placeholder='" + name + "' id='" + getId.call(this, index) + "'>" + name + "</button>";

}

/\*\*

\* [selectField displays a select field]

\* @param {string} name

\* @param {string} options

\* @param {number} index

\* @return {string}

\*/

function selectField(name, options, index) { fungsi menampilkan field yang telah dipilih

var html = "<select class='select-default' id='" + getId.call(this, index) + "'>";

options.forEach(function(option) {

html += "<option class='option-default' id='" + option + "'>" + option + "</option>";

});

html += "</select>"

return html;

}

/\*\*

\* [radioField displays a radio field]

\* @param {string} name

\* @param {string} options

\* @param {number} index

\* @return {string}

\* @private

\*/

function radioField(name, options, index) { fungsi menampilkan field tipe radio

var html = name ? "<p>choose the " + name + "</p>" : "";

firstOption = options.shift();

html += "<input checked type='radio' class='radio-default' id='" + getId.call(this, index) + "' value='" + firstOption + "'>" + firstOption + "</br> "

options.forEach(function(option) {

html += "<input type='radio' id='" + name + "' value='" + option + "'>" + option + "</br> "

});

return html;

}

/\*\*

\* [numberField displays a numberField]

\* @param {string} name

\* @param {number} min

\* @param {number} max

\* @param {number} index

\* @return {string}

\* @private

\*/

function numberField(name, min, max, index) { fungsi menampilkan penomoran jumlah field yang ada

return "<input class='input-default' placeholder='" + min + "-" + max + "' type='number' id='" + getId.call(this, index) + "' min='" + min + "' max='" + max + "' autofocus></input>"

}

/\*\*

\* [rangeField displays a range field]

\* @param {string} name

\* @param {number} min

\* @param {number} max

\* @param {number} index

\* @return {string}

\* @private

\*/

function rangeField(name, min, max, defaultValue, index) { fungsi membuat range pada field

var html = name ? "</br>" + name + "</br>" : "";

return html + "<input class='range-default' type='range' id='" + getId.call(this, index) + "' min='" + min + "' max='" + max + "' value='"+defaultValue+"'></input>";

}

/\*\*

\* [inputFields handles the display of all types of fields]

\* Returns a string of the fields' HTML

\* @return {string}

\* @private

\*/

function inputFields() { fungsi mengatur semua tipe display dalam field

var html = "";

var index = 0;

this.fields.forEach(function(field) {

if (field.type === 'text') {

html += textField.call(this, field.text, (field.id) ? field.id : index);

} else if (field.type === 'select') {

html += selectField.call(this, field.text, field.options, (field.id) ? field.id : index);

} else if (field.type === 'radio') {

html += radioField.call(this, field.text, field.options, (field.id) ? field.id : index);

} else if (field.type === 'number') {

html += numberField.call(this, field.text, field.min, field.max, (field.id) ? field.id : index);

} else if (field.type === 'range') {

html += rangeField.call(this, field.text, field.min, field.max, field.value, (field.id) ? field.id : index);

} else if (field.type === 'button') {

html += buttonField.call(this, field.text, (field.id) ? field.id : index);

} else if (field.type === 'basic-text') {

html += basicText.call(this, field.text);

}

index++;

}.bind(this));

return html;

}

/\*\*

\* [attachCallbacks attaches the proper callback functions to each form.]

\* @private

\*/

function attachCallbacks() { menempatkan fungsi callback yang sesuai pada masing-masing form

var index = 0;

this.fields.forEach(function(field) {

if (field.type === 'button') {

$("#" + getId.call(this, (field.id) ? field.id : index)).click(field.callback);

}

if(field.key){

// $("body").unbind();

// console.log(Mousetrap);

// $("body").keypress(function(event){

// if(event.which == field.key.charCodeAt()){

// console.log(field);

// field.callback();

// }

// });

}

index++;

}.bind(this));

}

}

* Complex object

\* Base class for complex objects. Objects which move on the map such as enemies will inherit this class.

\* @constructor

\*/

Class yang mengatur data dari objek yang kompleks seperti enemy yang bergerak.

var ComplexObject = function() {

this.animations;

this.currentDirection = "";

this.lastFrameDirection = "";

/\*\*

\* abstract method for updating movement animation

\* @param {number} deltax

\* @param {number} deltay

\* @abstract

\*/

this.updateMovementAnimation = function(deltax, deltay) {

};

};

* Person

\* Base class for players

\* @constructor

\*/

Class ini memiliki peran sebagai parent class yang menampung berbagai data maupun parameter mengenai player dalam game.

var Person = function() {

this.animations;

this.currentDirection = "";

this.lastFrameDirection = "";

this.scout = DEFAULT\_PERSON\_SIGHT\_DISTANCE;

this.hp = DEFAULT\_PERSON\_HEALTH;

this.attack = DEFAULT\_PERSON\_ATTACK;

this.healthBar;

/\*\*

\* Updates movement animation

\* @param {number} deltax change in x

\* @param {number} deltay change in y

\*/

this.updateMovementAnimation = function(deltax, deltay) {

if (deltax > 0 && this.lastFrameDirection !== "walk-right") {

this.animations.gotoAndPlay("walk-right");

this.lastFrameDirection = "walk-right";

} else if (deltax < 0 && this.lastFrameDirection !== "walk-left") {

this.animations.gotoAndPlay("walk-left");

this.lastFrameDirection = "walk-left";

} else if (deltay > 0 && this.lastFrameDirection !== "walk-front") {

this.animations.gotoAndPlay("walk-front");

this.lastFrameDirection = "walk-front";

} else if (deltay < 0 && this.lastFrameDirection !== "walk-back") {

this.animations.gotoAndPlay("walk-back");

this.lastFrameDirection = "walk-back";

}

};

};

* Simple object

\* Base class for all simple objects.

\* @constructor

\*/

Class yang menampung data maupun parameter dari objek yang simple seperti posisi, dan animasinya.

var SimpleObject = function(x, y) {

this.x = x; // mendefinisikan posisi x

this.y = y; // mendefinisikan posisi y

this.spriteSheet; // Sprite sheet untuk class simple object

this.animations; // Animasi diambil dari sprite sheet untuk di gambar dalam game

};

* Widget

\* Widget is a 'abstract class used to construct Dom based ui element classes and control their position on screen'

\* @param {Object} options object with x, y, container and el

\*/

Class Widget adalah kelas abstrak yang digunakan untuk membangun UI pada dom dan mengatur posisinya di layer

var Widget = function(options) { //function option dalam class widget

this.x = options.x || null;

this.y = options.y || null;

this.cssClass = options.cssClass || null;

this.cssId = options.cssId || null;

this.container = $('body');

this.el = {}; // mereset kode berskala kecil nantinya dalam game

console.log(this.cssClass);

/\*\*

\* Returns a string with position information in css form

\* @return {string}

\*/

this.getPositionCSS = function() {

if (!this.x && !this.y) {

return "";

} else {

return "style='position:absolute; top:" + this.y + "px; left: " + this.x + "px;'";

}

};

// ugly hack to have to move this declaration here, since we want to use the getPositionCSS method:

this.el = $("<div class='forms' " + this.getPositionCSS() + "></div>");

if(this.cssClass){

this.el.addClass(this.cssClass);

}else{

this.el.addClass("forms-default");

}

/\*\*

\* Hides element

\* @return {Widget}

\*/

this.hide = function() { fungsi menyembunyikan elemen class widget

this.el.hide();

return this

};

/\*\*

\* Shows element

\* @return {void} []

\*/

this.show = function() { fungsi menampilkan informasi yang ada dalam class widget

this.el.show();

};

/\*\*

\* Removes element

\* @return {void} []

\*/

this.destroy = function() { berperan sebagai fungsi yang menghilangkan suatu elemen.

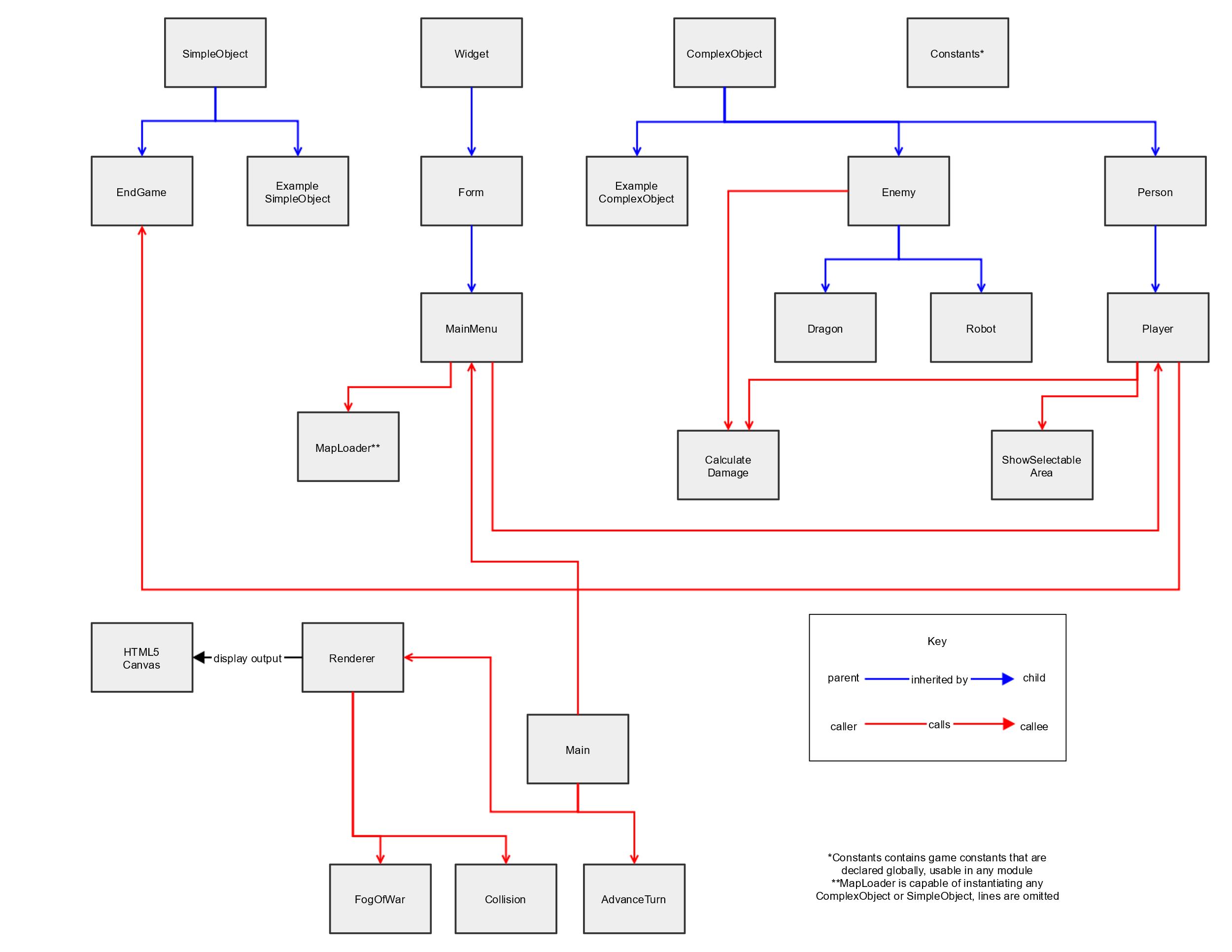
this.el.remove();

}

}

* Relasi antar Class

Relasi yang terjadi pada tiap kelas kebanyakan adalah inheritance. Meneruskan elemen yang ada pada class anak dibawahnya. Berikut gambar penjelasan

Tiap kelas yang telah disebutkan memiliki kelas child dibawahnya yang nantinya akan membawa elemen berupa fungsi, parameter, dan sebagainya hingga goal dalam game tercapai.