**CODE SMELLS**

**Speculative Generality**

net.sf.freecol.server.ai.AIColony.getWorkerWishes()

Probably they created this method thinking about a future version, but they do not use this method, so you could remove this code Smell by removing this method, because it is never used.

public List<WorkerWish> getWorkerWishes() {

return *transform*(wishes, w -> w instanceof WorkerWish,

w -> (WorkerWish)w);

}

**Long Method:**

net.sf.freecol.client.gui.panel.report.ReportCompactColonyPanel.updateColony(ColonySummary)

This method is too long, it could be better to divide him in sub-methods dividing the complexity to other methods, i.e. there the key specific for each functionality in the if else, they could made a specific method that handle this key for specific cases.

private void updateColony(ColonySummary s) {

final String cac = s.colony.getId();

final UnitType defaultUnitType

= spec.getDefaultUnitType(s.colony.getOwner());

List<JComponent> buttons = new ArrayList<>(16);

JButton b;

Color c;

StringTemplate t;

Building building;

// Field: A button for the colony.

// Colour: bonus in {-2,2} => {alarm, warn, plain, export, good}

// Font: Bold if famine is threatening.

c = (s.bonus <= -2) ? *cAlarm*

: (s.bonus == -1) ? *cWarn*

: (s.bonus == 0) ? *cPlain*

: (s.bonus == 1) ? *cExport*

: *cGood*;

String annotations = "", key;

t = StringTemplate.*label*(",");

if ((building = s.colony.getStockade()) == null) {

key = "annotation.unfortified";

t.add(Messages.*message*("report.colony.annotation.unfortified"));

} else {

key = "annotation." + building.getType().getSuffix();

t.add(Messages.*message*(building.getLabel()));

}

if (ResourceManager.*getStringResource*(key, false) != null) {

annotations += ResourceManager.*getString*(key);

}

if (!s.colony.getTile().isCoastland()) {

key = "annotation.inland";

t.add(Messages.*message*("report.colony.annotation.inland"));

} else if ((building = s.colony.getWorkLocationWithAbility(Ability.*PRODUCE\_IN\_WATER*, Building.class)) == null) {

key = "annotation.coastal";

t.add(Messages.*message*("report.colony.annotation.coastal"));

} else {

key = "annotation." + building.getType().getSuffix();

t.add(Messages.*message*(building.getLabel()));

}

if (ResourceManager.*getStringResource*(key, false) != null) {

annotations += ResourceManager.*getString*(key);

}

/\* Omit for now, too much detail.

for (GoodsType gt : spec.getLibertyGoodsTypeList()) {

if ((building = s.colony.getWorkLocationWithModifier(gt.getId(), Building.class)) != null) {

key = "annotation." + building.getType().getSuffix();

t.add(Messages.message(building.getLabel()));

if (ResourceManager.hasResource(key))

annotations += ResourceManager.getString(key);

}

}\*/

/\* Omit for now, too much detail.

for (GoodsType gt : spec.getImmigrationGoodsTypeList()) {

if ((building = s.colony.getWorkLocationWithModifier(gt.getId(), Building.class)) != null) {

key = "annotation." + building.getType().getSuffix();

t.add(Messages.message(building.getLabel()));

if (ResourceManager.hasResource(key))

annotations += ResourceManager.getString(key);

}

}\*/

/\* Font update needed

if ((building = s.colony.getWorkLocationWithAbility(Ability.TEACH, Building.class)) != null) {

key = "annotation." + building.getType().getSuffix();

t.add(Messages.message(building.getLabel()));

if (ResourceManager.hasResource(key)) annotations += ResourceManager.getString(key);

}\*/

if ((building = s.colony.getWorkLocationWithAbility(Ability.*EXPORT*, Building.class)) != null) {

annotations += "\*";

t.add(Messages.*message*(building.getLabel()));

}

b = newButton(cac, s.colony.getName() + annotations, null, c,

StringTemplate.*label*(": ").add(s.colony.getName())

.add(Messages.*message*(t)));

if (s.famine) b.setFont(b.getFont().deriveFont(Font.*BOLD*));

reportPanel.add(b, "newline");

// Field: Size

c = *cGood*;

t = *stpld*("report.colony.size");

reportPanel.add(newButton(cac, Integer.*toString*(s.unitCount), null, c, t));

// Field: The number of colonists that can be added to a

// colony without damaging the production bonus

if (s.unitsToAdd > 0) {

c = *cGood*;

t = *stpld*("report.colony.growing")

.addName("%colony%", s.colony.getName())

.addAmount("%amount%", s.unitsToAdd);

b = newButton(cac, Integer.*toString*(s.unitsToAdd), null, c, t);

} else {

b = null;

}

reportPanel.add((b == null) ? new JLabel() : b);

// Field: the number of colonists to remove to fix the inefficiency.

// Colour: Blue if efficient/Red if inefficient.

if (s.unitsToRemove > 0) {

c = s.bonus < 0 ? *cAlarm* : *cGood*;

t = *stpld*("report.colony.shrinking")

.addName("%colony%", s.colony.getName())

.addAmount("%amount%", s.unitsToRemove);

b = newButton(cac, Integer.*toString*(s.unitsToRemove), null, c, t);

} else {

b = null;

}

reportPanel.add((b == null) ? new JLabel() : b);

// Field: The number of potential colony tiles that need

// exploring.

// Colour: Always cAlarm

int n = *count*(s.tileSuggestions,

TileImprovementSuggestion::isExploration);

if (n > 0) {

t = *stpld*("report.colony.exploring")

.addName("%colony%", s.colony.getName())

.addAmount("%amount%", n);

b = newButton(cac, Integer.*toString*(n), null, *cAlarm*, t);

} else {

b = null;

}

reportPanel.add((b == null) ? new JLabel() : b);

// Fields: The number of existing colony tiles that would

// benefit from improvements.

// Colour: Always cAlarm

// Font: Bold if one of the tiles is the colony center.

for (TileImprovementType ti : spec.getTileImprovementTypeList()) {

if (ti.isNatural()) continue;

n = 0;

boolean center = false;

for (TileImprovementSuggestion tis : s.tileSuggestions) {

if (tis.tileImprovementType == ti) {

n++;

if (tis.tile == s.colony.getTile()) center = true;

}

}

if (n > 0) {

c = *cAlarm*;

if (n == 1) {

TileImprovementSuggestion tis = *first*(s.tileSuggestions);

if (*any*(tis.tile.getUnits(),

u -> (u.getState() == Unit.UnitState.*IMPROVING*

&& u.getWorkImprovement() != null

&& u.getWorkImprovement().getType()

== tis.tileImprovementType))) {

c = *cWarn*; // Work is underway

}

t = *stpld*("report.colony.tile." + ti.getSuffix()

+ ".specific")

.addName("%colony%", s.colony.getName())

.addStringTemplate("%location%",

tis.tile.getColonyTileLocationLabel(s.colony));

} else {

t = *stpld*("report.colony.tile." + ti.getSuffix())

.addName("%colony%", s.colony.getName())

.addAmount("%amount%", n);

}

b = newButton(cac, Integer.*toString*(n), null, c, t);

if (center) b.setFont(b.getFont().deriveFont(Font.*BOLD*));

} else {

b = null;

}

reportPanel.add((b == null) ? new JLabel() : b);

}

// Fields: The net production of each storable+non-trade-goods

// goods type.

// Colour: cAlarm if too low, cWarn if negative, empty if no

// production, cPlain if production balanced at zero,

// otherwise must be positive, wherein cExport

// if exported, cAlarm if too high, else cGood.

for (GoodsType gt : this.goodsTypes) {

final ColonySummary.GoodsProduction gp = s.production.get(gt);

switch (gp.status) {

case *FAIL*:

c = *cAlarm*;

t = *stpld*("report.colony.production.low")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", -gp.amount)

.addAmount("%turns%", gp.extra);

break;

case *BAD*:

c = *cWarn*;

t = *stpld*("report.colony.production")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount);

break;

case *NONE*:

c = null;

t = null;

break;

case *ZERO*:

c = *cPlain*;

t = *stpld*("report.colony.production")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount);

break;

case *GOOD*:

c = *cGood*;

t = *stpld*("report.colony.production")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount);

break;

case *EXPORT*:

c = *cExport*;

t = *stpld*("report.colony.production.export")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount)

.addAmount("%export%", gp.extra);

break;

case *EXCESS*:

c = *cWarn*;

t = *stpld*("report.colony.production.high")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount)

.addAmount("%turns%", gp.extra);

break;

case *OVERFLOW*:

c = *cAlarm*;

t = *stpld*("report.colony.production.waste")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount)

.addAmount("%waste%", gp.extra);

break;

case *PRODUCTION*:

c = *cWarn*;

t = *stpld*("report.colony.production.maxProduction")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount)

.addAmount("%more%", gp.extra);

break;

case *CONSUMPTION*:

c = *cWarn*;

t = *stpld*("report.colony.production.maxConsumption")

.addName("%colony%", s.colony.getName())

.addNamed("%goods%", gt)

.addAmount("%amount%", gp.amount)

.addAmount("%more%", gp.extra);

break;

default:

throw new IllegalStateException("Bogus status: " + gp.status);

}

reportPanel.add((c == null) ? new JLabel()

: newButton(cac, Integer.*toString*(gp.amount), null, c, t));

}

// Field: New colonist arrival or famine warning.

// Colour: cGood if arriving eventually, blank if not enough food

// to grow, cWarn if negative, cAlarm if famine soon.

if (s.newColonist > 0) {

t = *stpld*("report.colony.arriving")

.addName("%colony%", s.colony.getName())

.addNamed("%unit%", defaultUnitType)

.addAmount("%turns%", s.newColonist);

b = newButton(cac, Integer.*toString*(s.newColonist), null,

*cGood*, t);

} else if (s.newColonist < 0) {

c = (s.famine) ? *cAlarm* : *cWarn*;

t = *stpld*("report.colony.starving")

.addName("%colony%", s.colony.getName())

.addAmount("%turns%", -s.newColonist);

b = newButton(cac, Integer.*toString*(-s.newColonist), null,

c, t);

if (s.famine) b.setFont(b.getFont().deriveFont(Font.*BOLD*));

} else {

b = null;

}

reportPanel.add((b == null) ? new JLabel() : b);

// Field: What is currently being built (clickable if on the

// buildqueue) and the turns until it completes, including

// units being taught, or blank if nothing queued.

// Colour: cWarn if no construction is occurring, cGood with

// turns if completing, cAlarm with turns if will block, turns

// indicates when blocking occurs.

// Font: Bold if blocked right now.

final String qac = *BUILDQUEUE* + cac;

if (s.build != null) {

int turns = s.completeTurns;

String bname = Messages.*getName*(s.build);

if (turns == *UNDEFINED*) {

t = *stpld*("report.colony.making.noconstruction")

.addName("%colony%", s.colony.getName());

b = newButton(qac, bname, null, *cWarn*, t);

} else if (turns >= 0) {

t = *stpld*("report.colony.making.constructing")

.addName("%colony%", s.colony.getName())

.addNamed("%buildable%", s.build)

.addAmount("%turns%", turns);

b = newButton(qac, bname + " " + Integer.*toString*(turns), null,

*cGood*, t);

} else { // turns < 0

turns = -(turns + 1);

t = *stpld*("report.colony.making.blocking")

.addName("%colony%", s.colony.getName())

.addAmount("%amount%", s.needed.getAmount())

.addNamed("%goods%", s.needed.getType())

.addNamed("%buildable%", s.build)

.addAmount("%turns%", turns);

b = newButton(qac, bname + " " + Integer.*toString*(turns),

null, *cAlarm*, t);

if (turns == 0) b.setFont(b.getFont().deriveFont(Font.*BOLD*));

}

buttons.add(b);

}

// Field: What is being trained, including shadow units for vacant

// places.

// Colour: cAlarm if completion is blocked, otherwise cPlain.

int empty = 0;

Building school = s.colony.getWorkLocationWithAbility(Ability.*TEACH*,

Building.class);

if (school != null) empty = school.getType().getWorkPlaces();

for (Entry<Unit, Integer> e

: *mapEntriesByValue*(s.teachers, *descendingIntegerComparator*)) {

Unit u = e.getKey();

ImageIcon ii = new ImageIcon(this.lib.getTinyUnitImage(u));

if (e.getValue() <= 0) {

t = *stpld*("report.colony.making.noteach")

.addName("%colony%", s.colony.getName())

.addStringTemplate("%teacher%",

u.getLabel(Unit.UnitLabelType.*NATIONAL*));

b = newButton(cac, Integer.*toString*(0), ii, *cAlarm*, t);

} else {

t = *stpld*("report.colony.making.educating")

.addName("%colony%", s.colony.getName())

.addStringTemplate("%teacher%",

u.getLabel(Unit.UnitLabelType.*NATIONAL*))

.addAmount("%turns%", e.getValue());

b = newButton(cac, Integer.*toString*(e.getValue()), ii,

*cPlain*, t);

}

buttons.add(b);

empty--;

}

if (empty > 0) {

final ImageIcon emptyIcon

= new ImageIcon(this.lib.getTinyUnitTypeImage(defaultUnitType, true));

t = *stpld*("report.colony.making.educationVacancy")

.addName("%colony%", s.colony.getName())

.addAmount("%number%", empty);

for (; empty > 0; empty--) {

buttons.add(newButton(cac, "", emptyIcon, *cPlain*, t));

}

}

addTogether(buttons);

// Field: The units that could be upgraded, followed by the units

// that could be added.

if (s.improve.isEmpty() && s.want.isEmpty()) {

reportPanel.add(new JLabel());

} else {

buttons.clear();

buttons.addAll(unitButtons(s.improve, s.couldWork, s.colony));

buttons.add(new JLabel("/"));

// Prefer to suggest an improvement over and addition.

for (UnitType ut : s.improve.keySet()) s.want.remove(ut);

buttons.addAll(unitButtons(s.want, s.couldWork, s.colony));

addTogether(buttons);

}

// *TODO: notWorking?*

}

Data Class:

net.sf.freecol.server.ai.ColonyPlan.BuildPlan

This class only contains data and no real functionality, only the getter method. This class may have more functionalities inside, that are now in responsibility of other classes.

private static class BuildPlan {

public final BuildableType type;

public double weight;

public double support;

public double difficulty;

public BuildPlan(BuildableType type, double weight, double support) {

this.type = type;

this.weight = weight;

this.support = support;

this.difficulty = 1.0f;

}

public double getValue() {

return weight \* support / difficulty;

}