Programming with Java

Pyi Soe

05-05-2022

မာတိကာ

၁

Karel the Robot စက်ရုပ်ကားရဲလ်

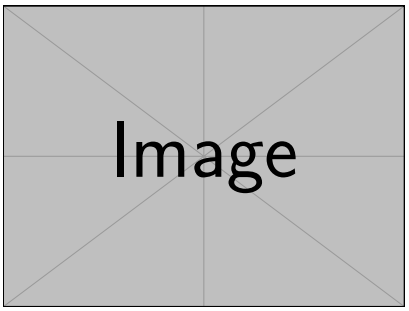
အသုံးပြုရတဲ့ **control flow statements** များတွင် `for`, `while`, `if`, `if else` တို့ပါဝင်သည်။ တွေကို ဒီအခန်းမှာ လေ့လာကြမယ်။ ကမ္ဘာ

အသုံးပြုရတဲ့ **control flow statements** များတွင် `for`, `while`, `if`, `if else` တို့ပါဝင်သည်။ တွေကို ဒီအခန်းမှာ လေ့လာကြမယ်။ ကမ္ဘာ

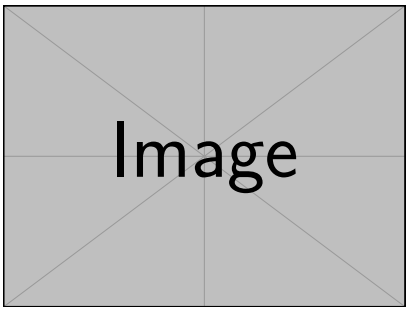
၁.၁ Commands Karel Understands ကားရဲလ်နားလည်သော ကွန်မန်းများ

၁.၁.၁ Commands Karel Understands ကားရဲလ်နားလည်သော ကွန်မန်းများ `for`, `while` `if`

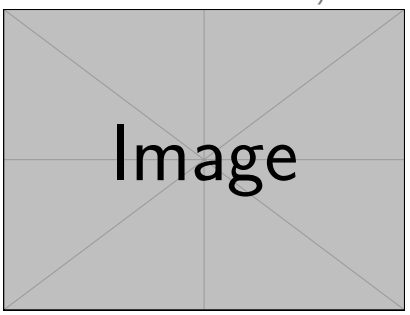
Commands Karel Understands ကားရဲလ်နားလည်သော ကွန်မန်းများ `for`, `while` `if`



က) Firts subfigure.

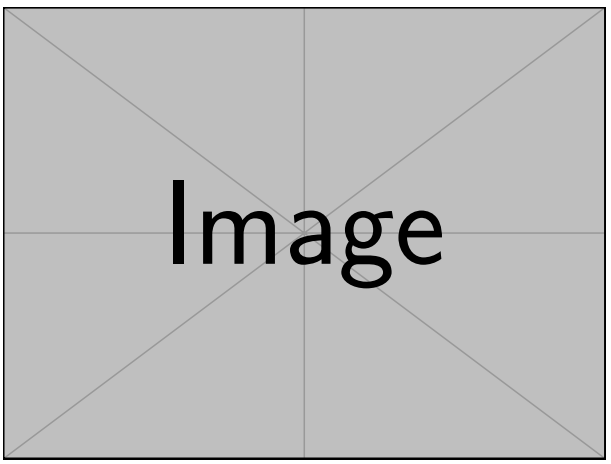


ခ) Second subfigure.

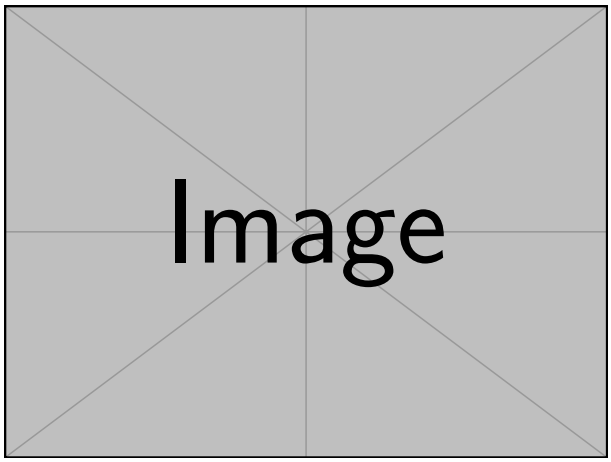


ဂ) Third subfigure.

ပုံ ၁.၁ – Creating subfigures in LaTeX.



ပုံ ၁.၂ – centered image



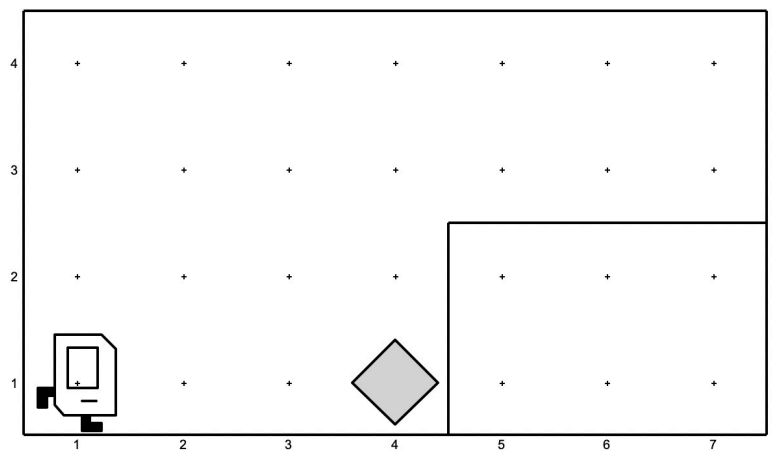
ပုံ ၁.၃ – your caption

J

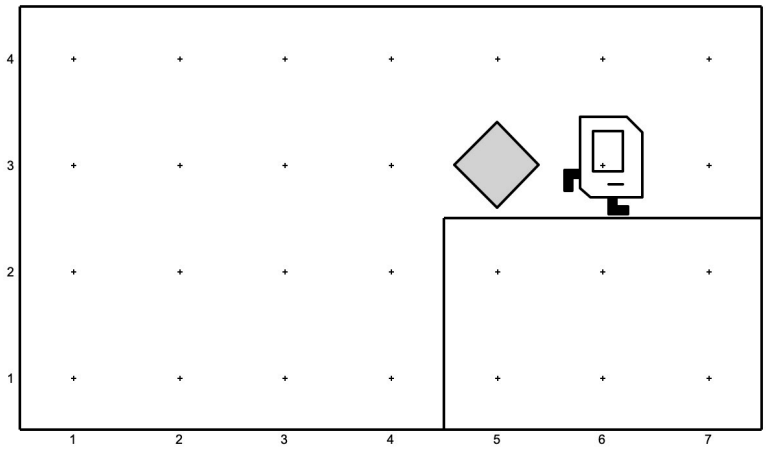
Programming with Karel the Robot

ပရိုဂရမ်းမင်း(programming) ဆိုတာဘာလဲရှင်းပြဖို့ စာတွေအများကြီးရေးနေတာထက် လက်တွေ့ ပရို ဂရမ်(program)လေးတွေ စရေးကြည့်ပြီးမှ ပရိုဂရမ်းမင်းဆိုတာ ဘာလဲ ပြောပြလိုက်ရင် ပိုပြီးနားလည် သဘောပေါက်လွယ်တယ်။

ပုံ ၂.၁ – ကားရဲလ်နှင့် ကားရဲလ်၏ကမ္ဘာ



ပုံ ၂.၂ – ဘီပါနေရာချွေပြီး



၂.၁ ကားရဲလ်နှင့် မိတ်ဆက်ခြင်း

ဒါ့ကြောင့် စက်ရုပ်ကားရဲလ်(Karel the Robot)ကို သူရောက်နေတဲ့ ကမ္ဘာမှာ အလုပ်တွေ လုပ်ခိုင်းဖို့ ပရိုဂရမ်လေးတွေ အရင်ဆုံး ရေးကြည့်ကြမယ်။ ပုံ ၂.၁ စာမျက်နှာ ၅ မှာ ပြထားတာက ကားရဲလ် ရောက်ရှိနေမယ့် နမူနာ ကမ္ဘာပါ။ ကားရဲလ်ရဲ့ ကမ္ဘာထဲက မီးခိုးရောင် ဆန္ဒင်းမကင်းကွက်ပုံ အရာကို ဘီပါ(beeper)လို့ ခေါ်တယ်။

ကားရဲလ်က စကားပြောပြီး ခိုင်းလို့မရဘူး။ သူနားလည်တဲ့ ကွန်မန်း(command)တွေကို ပရိုဂရမ်ရေးပြီးပဲ ခိုင်းလို့ရတယ်။

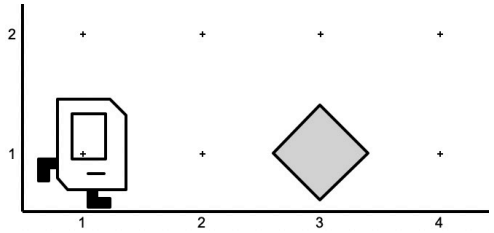
၂.၂ ကားရဲလ်နားလည်သော ကွန်မန်းလေးခု

ကားရဲလ်က အခြေခံအားဖြင့် ကွန်မန်းလေးခုကိုပဲ နားလည်တယ်။ `move`, `turnLeft`, `putBeeper`, `pickBeeper` ကွန်မန်းတို့ ဖြစ်တယ်။ `move` ကွန်မန်းက သူရပ်နေတဲ့ ကွန်နာကနေ ရှေ့တည့်တည့် ကပ်ရပ် ကွန်နာကို ရွှေ့ခိုင်းတာ။ ကွန်နာတွေကို အပေါင်းသင်္ကေတ လေးတွေနဲ့ ပြထားတယ်။ ပုံထဲမှာ သေးနေတဲ့ အတွက် အစက်လေးတွေလို့ ထင်ရတယ်။

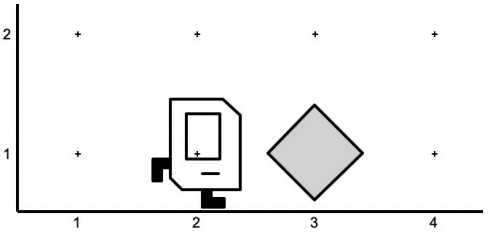
ကားရဲလ်ကို `putBeeper` ကွန်မန်းပေးလိုက်ရင်တော့ ကားရဲလ်က သူရှိနေတဲ့ ကွန်နာမှာ ဘီပါ(beeper)လို့ခေါ်တဲ့ အတုံးလေး တစ်ခုချထားလိမ့်မယ်။

`pickBeeper` ကွန်မန်းက ဘီပါကောက်ခိုင်းတာပါ။ ကားရဲလ်ရောက်နေတဲ့ ကွန်နာမှာ ဘီပါရှိရင် ဒီကွန်မန်းနဲ့ ကောက်ခိုင်းလို့ရတယ်။

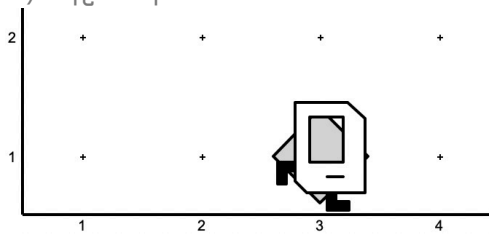
`turnLeft` ကွန်မန်းကတော့ ဘယ်ဘက်လှည့်ခိုင်းတာ။ မူလအနေအထားကနေ `turnLeft` ကွန်မန်း



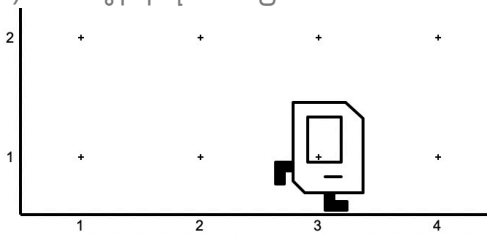
က) ကနဦး အနေအထား



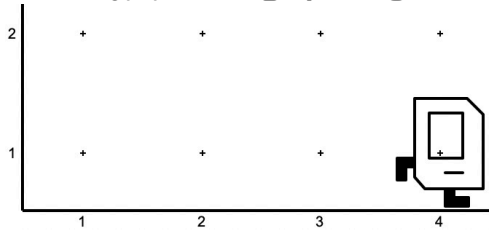
ခ) move ကွန်မန်းလုပ်ဆောင်ပြီး



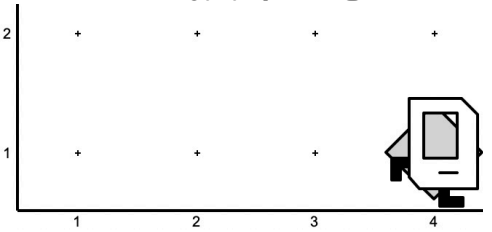
ဂ) move ကွန်မန်းတစ်ခါထပ်၍ လုပ်ဆောင်ပြီး



ဃ) pickBeeper ကွန်မန်းလုပ်ဆောင်ပြီး



c) move ကွန်မန်းတစ်ခါထပ်၍ လုပ်ဆောင်ပြီး



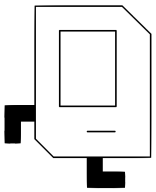
စ) putBeeper ကွန်မန်းလုပ်ဆောင်ပြီး

ပုံ ၂.၃ – move, pickBeeper, putBeeper ကွန်မန်းအသီးသီး အလုပ်လုပ်ပုံ

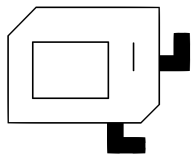
ပေးလိုက်တဲ့ အခါမှာ မျက်နှာမူရာ အရပ် ပြောင်းသွားပုံကို ပုံ ၂.၄ စာမျက်နှာ ၈ တွင်ကြည့်ပါ။ ညာဘက် လှည့်ခိုင်းဖို့ ကားရဲလ်ကို `turnRight` ကွန်မန်းနဲ့တော့ ခိုင်းလို့မရဘူး။ သူက `turnRight` ကွန်မန်းကို နား မလည်ပါဘူး။

၂.၃ ပထမဆုံး ကားရဲလ် ပရိုဂရမ်

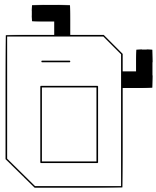
ဘီပါ(beeper)ကို မူလနေရာကနေ ပုံ ၂.၂ စာမျက်နှာ ၆ မှာပြထားတဲ့ နေရာကို ရွှေ့ခိုင်းရမှာပါ။ ကွန် မန်းတွေကို ဘယ်လို အစဉ်အတိုင်းပေးရမလဲ။ ဘီပါရှိတဲ့နေရာကိုသွားပြီး ဘီပါကောင်ခိုင်းဖို့က `move, move, move, pickBeeper` ကွန်မန်းပေးမယ်။ ပြီးတဲ့အခါ ရှေ့မှာနံရံရှိနေတယ်။ ဆက်သွားလို့မရတော့ ဘူး။ မြောက်ဘက်ကို သွားခိုင်းဖို့ `turnLeft`။ နှစ်ကြိမ်ထပ်ပြီးရွှေ့ခိုင်းရမယ်ဆိုတော့ `move, move`။ အရှေ့



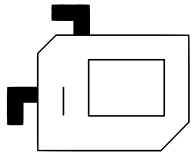
က) မူလ အနေအထား(အရှေ့ဘက်လှည့်)



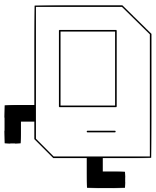
ခ) `turnLeft` တစ်ကြိမ်လုပ်ဆောင်ပြီး



ဂ) `turnLeft` တစ်ကြိမ် ထပ်၍လုပ်ဆောင်ပြီး



ဃ) `turnLeft` တစ်ကြိမ် ထပ်၍လုပ်ဆောင်ပြီး



င) `turnLeft` နောက်ဆုံးတစ်ကြိမ် လုပ်ဆောင်ပြီး

ပုံ ၂.၄ – ကနဦး အရှေ့ဘက် မျက်နှာမူနေရာမှ

ဘက် ပြန်လှည့်ခိုင်းဖို့ ညာဘက်လှည့်ခိုင်းရမယ်။ ဒါပေမယ့် `turnRight` ကွန်မန်းမရှိဘူး။ `turnLeft` သုံးခါလုပ်ခိုင်းလိုက်ရင်လည်း အရှေ့ဘက်လှည့်သွားမှာပဲ။ တစ်ခါထပ်ပြီး `move` ခိုင်းလိုက်ရင် ဘိပါကိုထားခိုင်းချင်တဲ့နေရာ ရောက်ပြီ။ `putBeeper` နဲ့ ဘိပါချထားခိုင်းလိုက်မယ်။ ဘိပါကိုမြင်သာအောင် `move` တစ်ခါထပ်လုပ်ခိုင်းလိုက်မယ်။

အစဉ်အတိုင်း ပေးရမယ့် ကွန်မန်းတွေက `move, move, move, pickBeeper, turnLeft, move, move, turnLeft, turnLeft, turnLeft, move, putBeeper, move` ဖြစ်မယ်။ ဒီအစီအစဉ်အတိုင်း ကွန်မန်းတွေကို ပရိုဂရမ်ရေးပေးရမယ်။ Java Programming Language ကို အသုံးပြုပြီး ရေးရမှာပါ။

ပရိုဂရမ်ရေးဖို့အတွက် ကွန်ပျူတာ(computer) နားလည်တဲ့ ဘာသာစကား တစ်မျိုးမျိုးကို အသုံးပြုပြီး ရေးရတယ်။ ကွန်ပျူတာ နားလည်တဲ့၊ ကွန်ပျူတာပရိုဂရမ်ရေးဖို့ အသုံးပြုတဲ့ ဘာသာစကားကို “ပရိုဂရမ်ရေးရန် အသုံးပြုသည့် ဘာသာစကား”(programming language) လို့ ခေါ်တယ်။

Listing ၂.၁ – ကားရဲလ် ပရိုဂရမ် template

```
public class ClassName extends stanford.karel.Karel {
        public void run() {
                //Karel commands
        }
}
```

မြန်မာ၊ အင်္ဂလိပ် စတဲ့ လူတွေရဲ့ ဘာသာစကားတွေမှာ သဒ္ဒါရှိသလိုပဲ programming language တွေမှာလည်း သဒ္ဒါရှိတယ်။ ပရိုဂရမ်ရေးတဲ့အခါ အသုံးပြုတဲ့ programming language ရဲ့ ရေးပုံရေးနည်း၊ စည်းမျဉ်းစည်းကမ်း သတ်မှတ်ချက်တွေကို လိုက်နာဖို့လိုတယ်။ programming language ရဲ့ သဒ္ဒါကို ဆင်းတက်စ်(syntax) လို့ ခေါ်တယ်။

ဒီအခန်းအတွက်က ဆင်းတက်စ်အသေးစိတ် နားလည်ဖို့ မလိုသေးဘူး။ Java Programming Language ရဲ့ လိုအပ်ချက်အရ ပုံစံချပေးထားတဲ့ ကားရဲလ် ပရိုဂရမ် template ကိုအသုံးပြုပြီး ကွန်မန်းတွေကို template ထဲမှာပဲဖြည့်ရေးမယ်။ template က အခုလိုပုံစံပါ။

`ClassName` နေရာမှာ ရေးမယ့် ပရိုဂရမ်အမည်ကို အစားထိုးရမယ်။ `MeetKarel` လို့ ပေးရအောင်။

```
public class MeetKarel extends stanford.karel.Karel {
        public void run() {
                //Karel commands
        }
}
```

ကွန်မန်းတွေကိုတော့ `public void run()` နဲ့ဆိုင်တဲ့ တွန့်ကွင်း အဖွင့်နှင့် အပိတ်အတွင်းမှာ ရေးပေးရမယ်။ ဖတ်ရလွယ်အောင် တစ်လိုင်းမှာ တစ်ခုပဲရေးလေ့ရှိတယ်။ `move` ကွန်မန်းကို Java Programming Language ဆင်းတက်စ်နဲ့ကိုက်ညီအောင် `move()` လို့ရေးဖို့ လိုအပ်တယ်။ ကွန်မန်းတစ်ခုအပြီးမှာ ဆမ်မီကော်လံ(semicolon) ထည့်ပေးဖို့လဲလိုတယ်။ ဝါကျတစ်ကြောင်းဆုံးရင် ပုဒ်မချပေးရတဲ့ သဘောပဲ။ ကွန်မန်းတစ်ခု ပြီးတိုင်း ဆမ်မီကော်လံ ထည့်ပေးဖို့လိုတယ်။

```
public class MeetKarel extends stanford.karel.Karel {
        public void run() {
                move();
        }
}
```

Listing ၂.၂ – ကားရဲလ် ပရိုဂရမ် template

```
public class MeetKarel extends stanford.karel.Karel {
        public void run() {
                move();
                move();
                move();
                pickBeeper();
                turnLeft();
                move();
                move();
                turnLeft();
                turnLeft();
                turnLeft();
                move();
                putBeeper();
                move();
        }
}
```

ဘိပါရွှေဖို့အတွက် ပရိုဂရမ်အပြည့်အစုံကို Listing ၂.၂ မှာ ကြည့်ပါ။ template ဖွဲ့စည်းတည်ဆောက်ထားပုံ၊ အသုံးပြုထားတဲ့ `public`, `class`, `extends`, `void` စတဲ့ စကားလုံးတွေရဲ့ အဓိပ္ပါယ် စတာတွေ နားလည်ဖို့ လိုနေသေးပေမယ့် လောလောဆယ်ဆယ်တော့ ဒါတွေခဏထားပြီး ကားရဲလ်ကို ခိုင်းထားတဲ့ ကွန်မန်းတွေဟာ ဘိပါကို နေရာမှန်အောင် ရွှေ့ပေးဖို့အတွက် မှန်ကန်လား၊ ပြဿနာတစ်ခုခု ရှိနေမလား သိရအောင် ပရိုဂရမ်ကို run ကြည့်ရအောင်။ ပရိုဂရမ်က လိုင်းအရေအတွက် နည်းပေမယ့် မှားနိုင်ပါတယ်။ ဘိပါရှိတဲ့ ကွန်နာ ကို မရောက် သေးပဲ ကောက်ခိုင်းတာ၊ အရပ်မျက်နှာ မှားပြီး လှည့်ခိုင်းမိတာ၊ အကြိမ်အရေအတွက် လိုနေတာ၊ ပိုသွားတာ စသည်ဖြင့် မှားနိုင်တဲ့ အချက်တွေ အများကြီးပါ။

၂.၄ IntelliJ

ပရိုဂရမ် ရေးဖို့ IntelliJ IDEAIDE ကို ဖွင့်။

၂.၄.၁ `class` သတ်မှတ်ခြင်း

`class` တစ်ခုသတ်မှတ်ဖို့အတွက်

Listing ၂.၃ – caption

```
public class MeetKarel extends stanford.karel.Karel{

}
```

Listing ၂.၄ – caption

```
public class MeetKarel extends stanford.karel.Karel{

}
```

၂.၄.၂ `class` သတ်မှတ်ခြင်း

`class` တစ်ခုသတ်မှတ်ဖို့အတွက်

၂.၅ Karel's World

ပုံ ၂.၁ စာမျက်နှာ ၅ အနောက်မှအရှေ့ ကွန်နာတွေ တလျှောက်က လမ်း(street)တွေ ဖြစ်ပြီး တောင်မှမြောက် ကွန်နာတွေ တလျှောက်က ရိပ်သာလမ်း(avenue)တွေ ဖြစ်တယ်။

ကွန်နာတွေက ရိပ်သာလမ်းနဲ့ လမ်းတွေဆုံတဲ့ လမ်းဆုံတွေ ဖြစ်တယ်။ ကွန်နာတစ်ခုကို ရည်ညွှန်းဖို့ ရိပ်သာလမ်းနဲ့ လမ်းနံပါတ်တွေကို အသုံးပြုမယ်။ ကားရဲလ်က (၁,၁) ကွန်နာ၊ ဘိပါက (၃,၁) ကွန်နာမှာ ရှိနေတယ်။ ပထမနံပါတ်က ရိပ်သာလမ်း၊ ဒုတိယက လမ်းနံပါတ်ပါ။ "ဘိပါက (၃,၁) ကွန်နာမှာရှိတယ်" လို့ ရေးရင် ဘိပါက နံပါတ် ၃ ရိပ်သာလမ်း နဲ့ ၁ လမ်းဆုံတဲ့ ကွန်နာမှာ ရှိနေတယ်လို့ ဆိုလိုတာ။

၂.၆ ကားရဲလ် ပရိုဂရမ်ကို ခွဲခြမ်းစိတ်ဖြာကြည့်ခြင်း

Listing ၂.၅ – language

```
class ClassName {

}
```

Listing ၂.၆ – language

```
public class MeetKarel{

}
```

Listing ၂.၇ – language

```
public class MeetKarel extends stanford.karel.Karel{

}
```

Listing ၂.၈ – language

```
import stanford.karel.Karel;
public class MeetKarel extends {

}
```

Listing ၂.၉ – language

```
public class MeetKarel extends stanford.karel.Karel{
        public void run(){

        }
}
```

၃

Control Flow ကွန်ထရိုးလ်ဖလိုး

ကားရဲလ်ကို ရှေ့ကို ၂၅ လှမ်း ရွှေ့ခိုင်းချင်တယ်။ `move(); move(); move(); ...move();` ၂၅ ခါ ရေးလို့တော့ရတာပေါ့။ ဒါပေမယ့် စာရိုက်ရတာ အချိန်လည်းကုန် လက်လဲညောင်း ဖြစ်မယ်။ `move();` ကို ၂၅ ကြိမ် ကျော့ပေးပါလို့ ခိုင်းလို့ရရင် ပိုမကောင်းဘူးလား။

ကားရဲလ်ရဲ့ ရှေ့တည့်တည့် ခပ်လှမ်းလှမ်းမှာ နံရံတစ်ခုရှိနေမယ်။ ဘယ်လောက်ဝေးလဲ မသိဘူးဆိုပါစို့။ ကားရဲလ်ကို နံရံဆီရောက်အောင် သွားခိုင်းချင်တယ်။ နံရံက ဘယ်လောက်ဝေးလဲ မသိတော့ `move` ကို ဘယ်နှစ်ကြိမ် ကျော့ခိုင်းရမလဲ မသိနိုင်ဘူး။ ရှေ့မှာရှင်းနေသေးသ၍ `move` ပါလို့သာ ခိုင်းလို့ရမယ်ဆိုရင် တော်တော်လေး အဆင်ပြေပြီ။

အခြေအနေတစ်ခု မှန်တော့မှပဲ ကွန်မန်းတွေကို လုပ်ဆောင်စေချင်တာမျိုးလဲ ရှိတယ်။ အဲဒီ အခြေနေနဲ့ မကိုက်ညီဘူး (တနည်းအားဖြင့် အခြေအနေက မှားနေခဲ့ရင်) ကွန်မန်းတွေကို မလုပ်ဆောင်ပဲ ကျော်သွားစေချင်တယ်။ ရှေ့မှာပြောခဲ့တဲ့ နံရံအောက်ခြေမှာ ဘိပါတစ်ခု ရှိနေနိုင်တယ်၊ ရှိချင်မှလည်း ရှိမယ်ဆိုပါစို့။ ဘိပါရှိနေခဲ့ရင် နံရံ အခြားတဘက်မှာ ဘိပါကို ထားခိုင်းချင်တယ်။ ဒါဆိုရင် ဘိပါရှိနေမှပဲ အခြားတဘက်ကို ရွှေ့ခိုင်းဖို့ လိုအပ်တဲ့ ကွန်မန်းတွေကို လုပ်ဆောင်စေချင်တယ်။ ဘိပါမရှိဘူးဆိုရင် အဲဒီ ကွန်မန်းတွေကို မလုပ်ဆောင်စေချင်ဘူး။

နောက်ထပ်တစ်မျိုးက အခြေအနေတစ်ခု မှန်ခဲ့ရင် လုပ်ဆောင်စေချင်တဲ့ ကွန်မန်းတွေနဲ့ မှားခဲ့ရင် လုပ်ဆောင်စေချင်တဲ့ ကွန်မန်းတွေကို မတူပဲဖြစ်နေတာမျိုးပါ။ တနည်းအားဖြင့် အခြေအနေပေါ် မူတည်ပြီး ခိုင်းရမယ့် အလုပ်ကမတူဘူး။ ဘိပါရှိခဲ့ရင် နံရံရဲ့ အခြားဘက်ကိုရွှေ့ခိုင်းချင်တယ်၊ မရှိခဲ့ရင်တော့ လာလမ်းအတိုင်း ပြန်လာစေချင်တယ် ဆိုပါစို့။ ဒါဆိုရင် ဘိပါရှိခြင်း၊ မရှိခြင်းပေါ် မူတည်ပြီး လုပ်ဆောင်ရမယ့် ကွန်မန်းတွေက မတူဘူး။

ဒီအခန်းမှာတော့ အထက်ပါ လိုအပ်ချက်မျိုးတွေအတွက် အသုံးပြုတဲ့ control flow statements တွေကို လေ့လာကြမယ်။ ပြန်ကျော့ဖို့အတွက် သုံးတဲ့ `for` loop နဲ့ `while` loop ကို အရင်ကြည့်ကြရအောင်။

၃.၁ for loop

`for` loop ကို ကွန်မန်းတစ်ခု သို့ တစ်စုကို ပြန်ကျော့ဖို့ အသုံးပြုနိုင်တယ်။ ဆင်းတက်စ်က အခုလိုပုံစံမျိုး နဲ့ ရေးရတယ်

```
for (int i = 0; i < N; i++) {
    //one or more commands here
}
```

`move` ကို နှစ်ဆယ့်ငါးကြိမ် ကျော့ချင်ရင်

```
for (int i = 0; i < 25; i++) {
    move();
}
```

ပြန်ကျော့ချင်တဲ့ ကွန်မန်းတွေကို တွန့်ကွင်း အဖွင့်နှင့် အပိတ်အတွင်း လိုင်းတွေမှာရေးပြီး `N` နေရာမှာ အကြိမ်အရေအတွက်ကို အစားထိုးပေးရမယ်။ `int i` မှာ ခြားထားရပါမယ်။ `inti` ဆိုရင် ဆင်းတက်စ် အမှားဖြစ်မယ်။ `i++` က တဆက်ထဲ ဖြစ်ရမယ်။ `i ++` သို့ `i + +` သို့ `i+ +` ရေးလို့ မရဘူး။ ကျန်တဲ့ စပေ့စ်တွေက ခြားသည်ဖြစ်စေ မခြားသည်ဖြစ်စေ ပြဿနာ ဆင်းတက်စ် အရတော့ မှန်တယ်။ ဒီလိုရေးလို့ ရပေမယ့် ဖတ်ရတာ ခက်တယ်။

```
for (int i=0;i<25;i++){
move();
}
```

ပူးကပ်နေပြီး အမြင်အရလည်း မရှင်းလင်းဘူး။

၃.၁.၁ for loop အသုံးပြုသည့် ဥပမာများ

ကားရဲလ်က ပုံ ၃.၁ စာမျက်နှာ ၁၅ ကမ္ဘာထဲမှာရှိနေမယ်။ လမ်းတလျှောက်လုံး ကွန်နာတစ်ခု စီတိုင်းမှာ ဘီပါတစ်ခု ချထားခိုင်းချင်တယ်။ ရိပ်သာလမ်းအရေအတွက်က ဒီပုစ္ဆာအတွက် အပြောင်းအလဲမရှိဘူး။ ပုံသေ ၅ ခုပဲဖြစ်မယ်လို့ ယူဆပါ။ ဘီပါချလိုက် ရှေ့ကွန်နာကို ရွှေ့လိုက်၊ ဘီပါချလိုက် ရှေ့ကွန်နာကို ရွှေ့လိုက် လုပ်ခိုင်းရမှာပေါ့။ ပြန်ကျော့ပေးရမယ့် ကွန်မန်းနှစ်ခုက `putBeeper` နဲ့ `move`။ ဘယ်နှစ်ကြိမ် ကျော့ခိုင်းရမှာလဲ။ သေချာဖို့လိုတယ်။ သေချာတာက ကားရဲလ်ရှေ့မှာ ကွန်နာလေးခုပဲရှိတဲ့အတွက် ရှေ့ကို လေးနေရာပဲ ရွှေ့လို့ရမယ်။ `N` နေရာမှာ `4` ထည့်ပြီး အခုလို ရေးရမယ်။ Listing ၃.၁ စာမျက်နှာ ၁၅ တွင်ကြည့်ပါ။

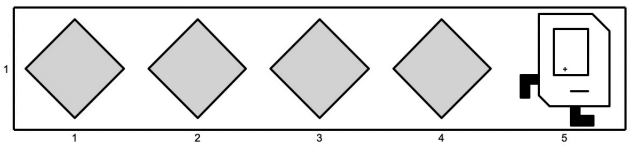
ပုံ ၃.၁ – ကားရဲလ်နှင့် ကားရဲလ်၏ကမ္ဘာ

Listing ၃.၁ – ပထမစမ်းကြည့်ပုံ

```
public class MakeRowOfFiveBeepers extends stanford.karel.Karel{
    public void run(){
            for(int i = 0; i < 4; i++) {
                    putBeeper();
                    move();
            }
    }
}
```

ဒီတိုင်း run ကြည့်တဲ့ရအောင်။ ရလဒ်ကို ပုံ ၃.၂ တွင်ကြည့်ပါ။ နောက်ဆုံး ကွန်နာမှာ ဘိပါချဖို့ကျန်နေတာ တွေ့ရမယ်။ ဘာကြောင့်ကျန်နေရတာလဲ။ သေချာ နားလည်အောင် အီထရေးရှင်းတစ်ခါပြီးတိုင်း ရှိနေမယ့် အခြေအနေကို ၃.၃ စာမျက်နှာ ၁၆ မှာကြည့်ပါ။ နောက်ဆုံး ကွန်နာမှာ ဘိပါချပေးဖို့အတွက် `for` loop အပိတ်တွန့်ကွင်း နောက်တစ်လိုင်းမှာ `putBeeper();` ကွန်မန်းရေးပေးရမယ်။ `for` loop ဘော်ဒီ အပြင်မှာရှိတဲ့အတွက် ပြန်ကျော့မယ့် ကွန်မန်း ထဲမှာ မပါဘူး။ နောက်ဆုံးမှာ တစ်ကြိမ်ပဲ လုပ်ဆောင်တယ်။ Listing ၃.၂ စာမျက်နှာ ၁၇ တွင်ကြည့်ပါ။

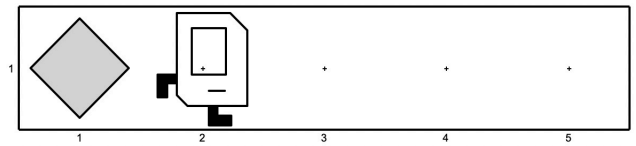
ပုံ ၃.၂ – ကားရဲလ်နှင့် ကားရဲလ်၏ကမ္ဘာ



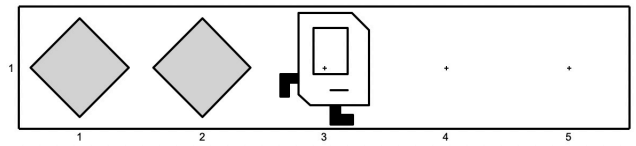
ပုံ ၃.၄ က) စာမျက်နှာ ၁၈ မှ စတုရန်းပုံ ကမ္ဘာထဲမှာ ကွန်နာတွေမှာ ဘိပါတစ်ခုစီ ချဖို့ `for` loop ကို သုံးနိုင်တယ်။ Listing ၃.၃ စာမျက်နှာ ၁၇ တွင်ကြည့်ပါ။ အီထရေးရှင်းတစ်ခါတိုင်းမှာ

```
putBeeper();
move();
turnLeft();
```

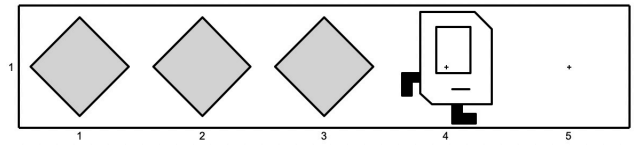
က) မူလ အနေအထား



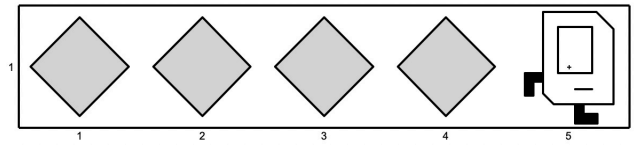
ခ) ပထမ တစ်ကျော့ပြီး



ဂ) ဒုတိယ တစ်ကျော့ပြီး



ဃ) တတိယ တစ်ကျော့ပြီး



င) စတုတ္ထမြောက် ကျော့ပြီး

ပုံ ၃.၃ – အီထရေးရှင်းတစ်ခုပြီး

ကို လုပ်ဆောင်မှာဖြစ်တယ်။ တစ်ကြိမ်ကျော့အပြီးမှာ ရှိနေမယ့် အခြေအနေကို ပုံ ၃.၄ စာမျက်နှာ ၁၈ တွင် ကြည့်ပါ။

၃.၂ `while` loop

အကြိမ်အရေအတွက် မသိပဲ အခြေအနေတစ်ခု မှန်နေသ၍ ပြန်ကျော့ဖို့ `while` loop ကို သုံးတယ်။ ဆင်းတက်စ်က အောက်ပါအတိုင်းရေးတယ်။

Listing ၃.၂ – ပထမစမ်းကြည့်ပုံ

```
public class MakeRowOfFiveBeepers extends stanford.karel.Karel{
    public void run(){
            for(int i = 0; i < 4; i++) {
                    putBeeper();
                    move();
            }

            putBeeper();
    }
}
```

Listing ၃.၃ – စတုရန်းပုံ ဘိပါ

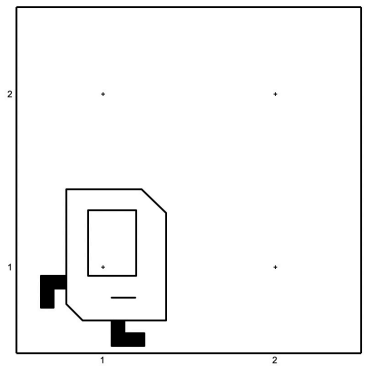
```
public class MakeBeeperSquare extends stanford.karel.Karel{
        public void run(){
                for(int i = 0; i < 4; i++) {
                        putBeeper();
                        move();
                        turnLeft();
                }
        }
}
```

```
while (CONDITION) {
    //commands to be executed while condition is true
}
```

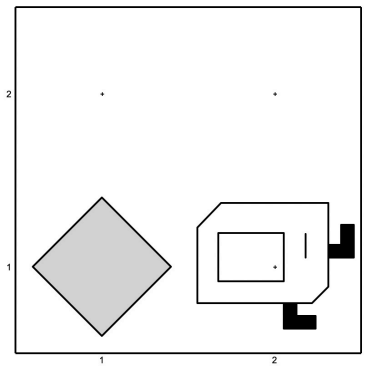
condition နေရာမှာ ၃.၁ စာမျက်နှာ ၂၄ ပါ ကွန်ဒီးရှင်းတစ်ခုကို အစားထိုးပေရမှာပါ။ ရှေ့မှာ ရှင်းနေသ၍ ရွှေ့ခိုင်းချင်တယ် ဆိုရင်

```
while (frontIsClear()) {
    move();
}
```

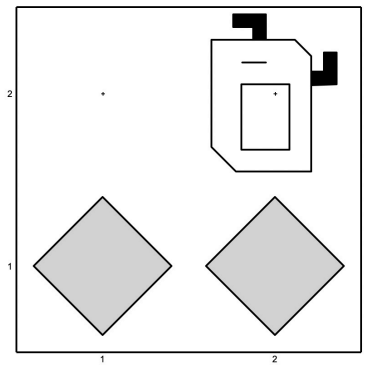
ပုံ ၃.၄ – Beeper Seqare



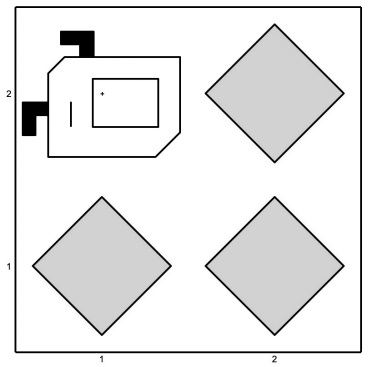
က) ကနဦး အနေအထား



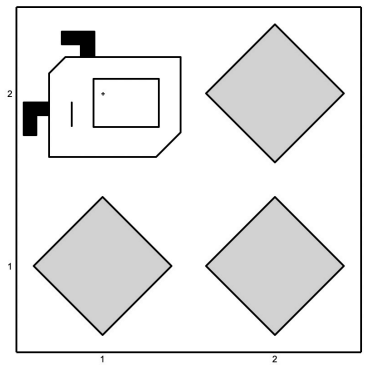
ခ) move ကွန်မန်းလုပ်ဆောင်ပြီး



ဂ) ကနဦး အနေအထား



ဃ) move ကွန်မန်းလုပ်ဆောင်ပြီး

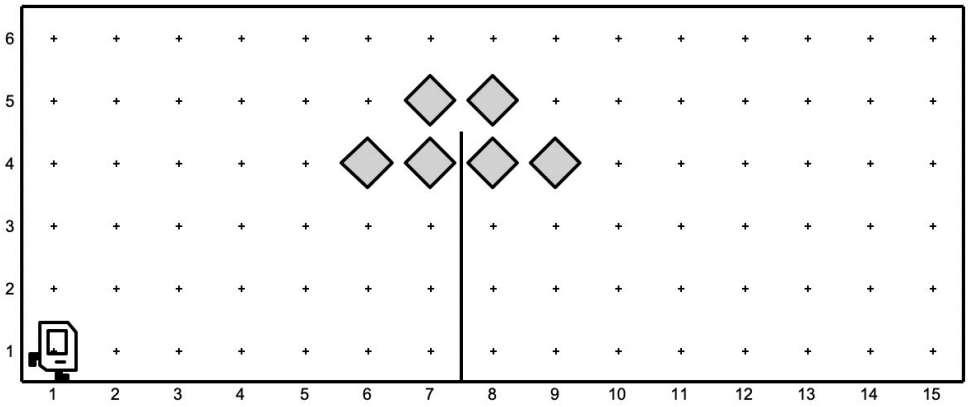


င) move ကွန်မန်းလုပ်ဆောင်ပြီး

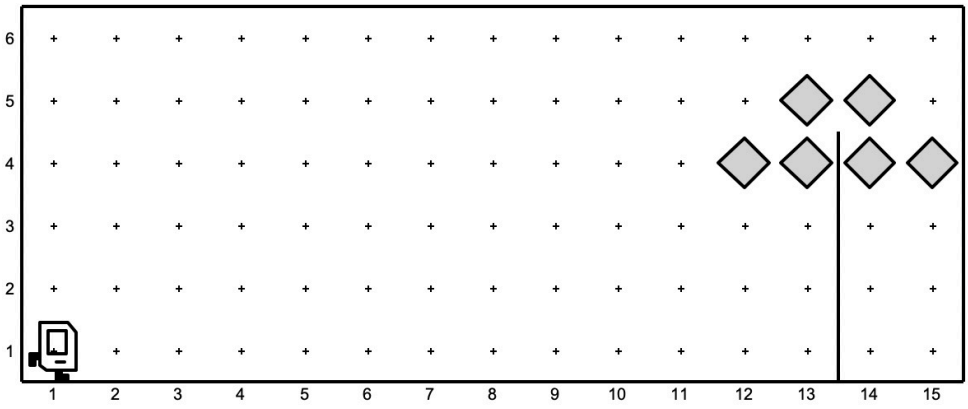
၃.၂.၁ `while` loop အသုံးပြုသည့် ဥပမာများ

ကားရဲလ် အရှေ့ဘက်မှာ အုန်းပင်တစ်ပင်ရှိတယ်။ ဘယ်လောက်အကွာအဝေးမှာ ရှိနေမလဲ ပုံသေပြောလို့မရဘူး။ နမူနာ နှစ်ခုကို ပုံမှာပြထားတယ်။ က မှာရှိနေသည်ဖြစ်စေ၊ ခ မှာရှိနေသည်ဖြစ်စေ၊ နံရံဆီကို သွားခိုင်းရမယ်။ ဒါ့အပြင် နောက်ထပ် အလားတူတဲ့ အခြားကမ္ဘာတွေထဲမှာလည်း နံရံက ဘယ်လောက်အကွာအဝေးမှာ ရှိနေသည်ဖြစ်စေ ကားရဲလ်ကို ရောက်အောင် သွားခိုင်းချင်တယ်။

ပုံ ၃.၅ – CococonutTree



က)



ခ)

နောက်ထပ် ဥပမာတစ်ခု ကြည့်ရအောင်။ ကားရဲလ်ကို လမ်းတလျှောက် ဘိပါတွေချထားခိုင်းချင်တယ်။ လမ်းအရှည်က ဘယ်လောက်ဖြစ်ဖြစ် အပြည့်ချထားပေးရမယ်။ လမ်းအရှည်က ပုံသေမဟုတ်တဲ့အတွက် ရှေ့

Listing ၃.၄ – CococonutTree.java

```
public class CococonutTree extends stanford.karel.Karel{
    public void run(){
            while(frontIsClear()){
                    move();
            }
    }
}
```

Listing ၃.၅ – MakeBeeperRow.java

```
public class MakeBeeperRow extends stanford.karel.Karel{
    public void run(){
            while(frontIsClear()){
                    putBeeper();
                    move();
            }
            putBeeper();
    }
}
```

မှာရှင်းနေသ၍ ဘိပါချလိုက် ရှေ့တိုးလိုက် ထပ်ခါထပ်ခါ လုပ်ခိုင်းရမှာပေါ့။

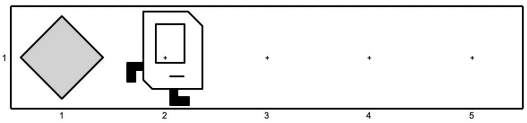
for loop မှာလိုပဲ loop ဘော်ဒီကို နောက်ဆုံးအကြိမ် လုပ်ဆောင်အပြီးမှာ ဘိပါချဖို့ကျန်နေမယ်။ ဒါကြောင့် while loop ဘော်ဒီ နောက်တစ်လိုင်းမှာ putBeeper(); ပါရမှာဖြစ်တယ်။

while loop အလုပ်လုပ်ပုံက အခုလိုပါ။ ကွန်ဒီးရှင်းကို အရင်ဆုံး စစ်ပါတယ်။ မှန်လျှင် ဘော်ဒီထဲက ကွန်မန်းတွေကို လုပ်ဆောင်တယ်။ တစ်ကြိမ်ကျော့ပြီးတိုင်း ကွန်ဒီးရှင်းကိုပြန်စစ်ပါတယ်။ မှန်နေသေးလျှင် ဘော်ဒီထဲက ကွန်မန်းတွေကို ထပ်ပြီးကျော့ပေးပါတယ်။ ဒီလိုမျိုး ကွန်ဒီးရှင်းစစ်လိုက်၊ မှန်နေရင် ပြန်ကျော့လိုက်ကို ထပ်ခါထပ်ခါ လုပ်နေပြီး ကွန်ဒီးရှင်းစစ်လိုက်လို့ မှားနေတဲ့ အခါကျတော့မှ ပြန်ကျော့တာကို ရပ်လိုက်တာဖြစ်တယ်။ ပြန်ကျော့တာရပ်လိုက်တာကို loop ထဲကနေထွက်သွား(loop exits) တယ်လို့ ပြောလေ့ရှိတယ်။

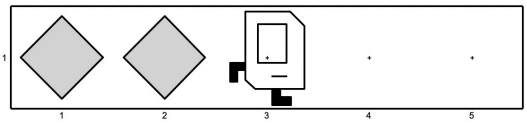
loop ထဲကနေထွက်သွား သွားပြီးနောက် loop ဘော်ဒီ အောက်ကမှာရှိတဲ့လိုင်းတွေကို ဆက်ပြီး လုပ်ဆောင်မှာ ဖြစ်တယ်။ ရှေ့ကဥပမာမှာ လေးကြိမ်မြောက်ထိ frontIsClear ကွန်ဒီးရှင်းက မှန်နေတယ်။ ဒါကြောင့် လေးခါကျော့ ခံရမယ်။ ငါးကြိမ်မြောက် ကွန်ဒီးရှင်းစစ်တဲ့အခါမှာတော့ frontIsClear ကမှားနေပြီ။ ရှေ့မှာ နံရံပိတ်နေတယ်။ ထပ်မကျော့တော့ပဲ loop ထဲကနေထွက်သွား ပြီး နောက်တလိုင်းက put-Beeper(); ကို ဆက်လက်လုပ်ဆောင်တယ်။

ပုံ ၃.၆ – MakeBeeperRow

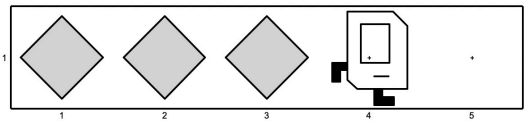
က)



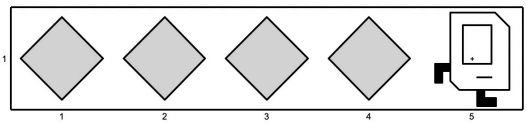
ခ)



ဂ)



ဃ)



င)

`while` loop ကွန်ဒီးရှင်းက ဘယ်တော့မှ မမှားတော့ပဲ အမြဲမှန်နေတာမျိုးရှိနိုင်တယ်။ ဒီအခါမှာတော့ loop ထဲကနေ ဘယ်တော့မှ ထွက်မသွားတော့ပဲ ဘော်ဒီထဲက ကွန်မန်းတွေကို ထာဝရ လုပ်ဆောင်နေတော့မှာ ဖြစ်တယ်။ ဒီလို အစဉ်အမြဲ ပြန်ကျော့နေမယ့် loop ကို infinite loop ဘော်ဒီ လို့ခေါ်ပါတယ်။ infinite loop ဘော်ဒီ ထဲကနေ မထွက်တော့တဲ့ အတွက် loop ဘော်ဒီ အောက် လိုင်းတွေမှာရှိတဲ့ ကွန်မန်းတွေကို လည်း လုပ်ဆောင်မှာ မဟုတ်ပါဘူး။

Listing ၃.၆ စာမျက်နှာ ၂၂ မှာ `pickBeeper();` ကို ဘယ်တော့မှ လုပ်ဆောင်ဖြစ်မှာ မဟုတ်ပါဘူး။

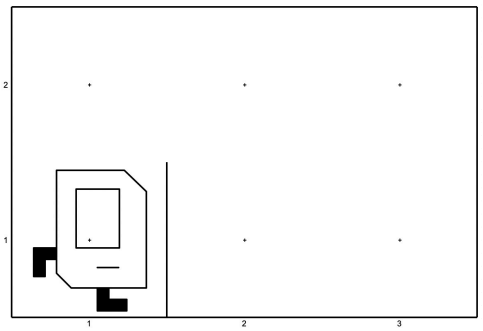
`while` loop ကွန်ဒီးရှင်းcondition‌က ပထမဆုံး စစစ်လိုက်တဲ့ အခါမှာပဲ မှားနေရင် ကွန်မန်းတွေကို တစ်ခါမှ မကျော့ပေးတော့ပဲ loop ထဲကနေထွက်သွား မှာဖြစ်တယ်။ ဒါကိုတော့ “loop ထဲကို လုံးဝ မဝင်

Listing ၃.၆ – GoAroundForever.java

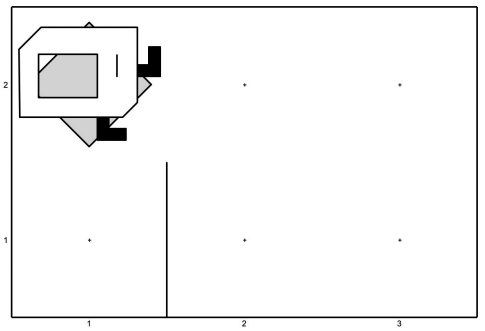
```
public class GoAroundForever extends stanford.karel.Karel{
    public void run(){
            //Karel will never get out of this loop
            while(frontIsClear()) {
                    move();
                    turnLeft();
            }
            //This command will never be executed
            pickBeeper();
    }
}
```

ဘူး"(loop is never entered) လို့ပြောလေ့ရှိတယ်။

ပုံ ၃.၇ – LoopNeverEntered



က)



ခ)

၃.၃ if statement

if statement ကို ကွန်မန်းတွေကို အခြေအနေမှန်တော့မှ လုပ်ဆောင်စေချင်တဲ့ အခါသုံးတယ်။ ဆင်းတက်စ်ကတော့ အခုလိုပုံစံ။

```
if ( CONDITION ) {
        // commands to be executed if CONDITION is true
}
```

Listing ၃.၇ – LoopNeverEntered.java A

```
public class LoopNeverEntered extends stanford.karel.Karel{
    public void run(){
            // This loop is never entered for default world
            // and its body will be totally skipped
            while(frontIsClear()) {
                    move();
                    turnLeft();
                    putBeeper();
            }
            // Commands below will be executed as usual
            turnLeft();
            move();
            putBeeper();
    }
}
```

CONDITION နေရာမှာ ဇယား ၃.၁ စာမျက်နှာ ၂၄ ထဲက ကွန်ဒီးရှင်းတစ်ခုကို အစားထိုးရမှာပါ။ လက်တွေ့ ဥပမာတစ်ခုကြည့်ရအောင်။ ကားရဲလ်က

၃.၄ if else statement

`if else` statement ကိုတော့ အခြေအနေတစ်ရပ် မှန်လျှင် လုပ်ဆောင်ချင်တာနဲ့ မှားလျှင် လုပ်ဆောင်ချင်တာ က မတူပဲကွဲပြား နေတဲ့အခါ သုံးတယ်။ ဆင်းတက်စ်ကတော့ အခုလိုပုံစံ။

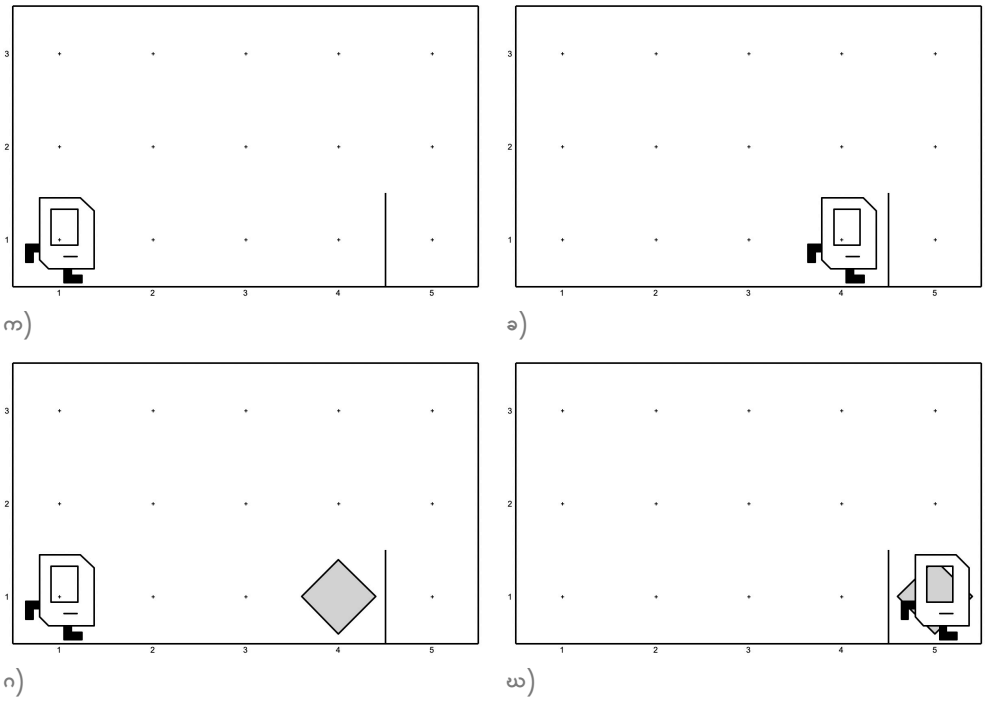
```
if ( CONDITION ) {
        // commands to be executed if CONDITION is true
} else {
        // commands to be executed if CONDITION is false
}
```

CONDITION နေရာမှာ ဇယား ၃.၁ စာမျက်နှာ ၂၄ ထဲက ကွန်ဒီးရှင်းတစ်ခုကို အစားထိုးရမှာပါ။ ရှေ့မှာတွေ့ခဲ့ တဲ့ ဥပမာမှာလိုပဲ ဘိပါရှိရင် ရွှေ့ပေးရမယ်၊ မရှိရင်တော့ စထွက်တဲ့နေရာကို ပြန်လာခိုင်းရမယ်ဆိုပါစို့။ ဒါဆို ရင် ဘိပါရှိရင် ရွှေ့ရမယ်၊ မရှိခဲ့ရင် (တနည်းအားဖြင့် beepersPresent ကွန်ဒီးရှင်းကမှားခဲ့လျှင်) ပြန်လာ ရမှာဖြစ်တယ်။

Karel Conditions	
`frontIsClear()`	`frontIsBlocked()`
`leftIsClear()`	`leftIsBlocked()`
`rightIsClear()`	`rightIsBlocked()`
`beepersPresent()`	`noBeepersPresent()`
`beepersInBag()`	`noBeepersInBag()`
`facingNorth()`	`notFacingNorth()`
`facingEast()`	`notFacingEast()`
`facingSouth()`	`notFacingSouth()`
`facingWest()`	`notFacingWest()`

Table ၃.၁ – ကားရဲလ်နားလည်သော ကွန်ဒီးရှင်းများ

ပုံ ၃.၈ – MooveMoveBeeperToOtherSide



က)

ခ)

ဂ)

ဃ)

Listing ၃.၈ – MoveBeeperToOtherSide.java

```
public class MoveBeeperToOtherSide extends stanford.karel.Karel {
    public void run() {

            // Code to go pick beeper

            // Check if beeper is there and move to the other side
            if (beepersPresent()) {
                    pickBeeper();
                    turnLeft();
                    move();
                    turnRight();
                    move();
                    turnRight();
                    move();
                    putBeeper();
                    turnLeft();
            }
    }

    // turnRight method definition
}
```

Listing ২.৫ – MoveBeeperToOtherSideOrComeBack.java

```
public class MoveBeeperToOtherSideOrComeBack extends stanford.karel.Karel {
	public void run() {

		// ...

		if (beepersPresent()) {
			pickBeeper();
			turnLeft();
			move();
			turnRight();
			move();
			turnRight();
			move();
			putBeeper();
			turnLeft();
		} else {
			turnLeft();
			turnLeft();
			move();
			move();
			move();
		}
	}

	// . . .turnRight method definition here

}
```

၄

Program Design - Solving More Complex Programs

ကားရဲလ်ကို ခိုင်းချင်တဲ့ အလုပ်တွေက ရှုပ်ထွေးလာတာနဲ့ အမျှ ပရိုဂရမ်ရေးရတာ ပိုခက်ခဲ လာတယ်။ ဘယ်လောက်ပဲ ရှုပ်ထွေးခက်ခဲတဲ့ အလုပ်ပဲဖြစ်ပါစေ၊ ရိုးရှင်းတဲ့ အလုပ်လေးတွေနဲ့ ဖွဲ့စည်းထာတာပါပဲ။ ဒါကြောင့် ခက်ခဲတဲ့အလုပ်ကို ရိုးရှင်းတဲ့ အလုပ်တွေဖြစ်အောင် ခွဲထုတ်ပြီး၊ တစ်ပိုင်းချင်းစီ ဖြေရှင်းသွားမယ် ဆိုရင် ပိုပြီးလွယ်ကူတယ်။ ဥပမာတစ်ခုကြည့်ရအောင်။

၄.၁ Top-down

ကားရဲလ်ကို ပုံ ၄.၁ စာမျက်နှာ ၂၈ မှာပြထားတဲ့ ဘီပါ(သတင်းစာဟု ယူဆပါ) ကို သွားကောက်ခိုင်းမယ်။ ကွန်မန်းတစ်ခုချင်း အစီအစဉ်ကို အသေးစိတ် အတိအကျ မစဉ်းစားသေးပဲ ပထမ တဆင့်မှာ လုပ်ခိုင်းရမယ့် အလုပ်တွေကို စဉ်းစားကြည့်လျှင်

(က) သတင်းစာရှိတဲ့ နေရာကိုသွား

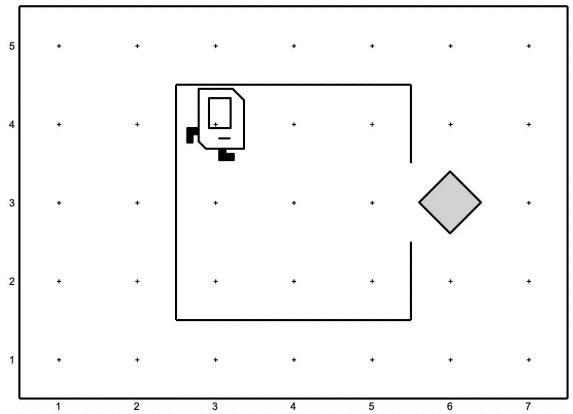
(ခ) သတင်းစာကောက်

(ဂ) နကိုမူလနေရာကို ပြန်လာ

ဆိုတဲ့ အလုပ် သုံးခု တွေ့ရမယ်။ "(က) သတင်းစာရှိတဲ့ နေရာကိုသွား" အတွက် လုပ်ရမယ့် အလုပ်တွေကို နောက်ထပ်တဆင့် ထပ်ခွဲထုတ်ကြည့်ရင်

- နံရံဆီကိုသွား

ပုံ ၄.၁ – ကားရဲလ်နှင့် ကားရဲလ်၏ကမ္ဘာ



- ညာဘက်လှည့်
- ရှေ့တိုး
- ဘယ်ဘက်လှည့်
- ရှေ့တိုး

“(ခ) စာမျက်နှာ ၂၇သတင်းစာကောက်” အတွက် အလုပ်တွေကို ထပ်ခွဲထုတ်ကြည့်ရင်

- သတင်းစာရှိ/မရှိ စစ်
- ရှိလျှင်ကောက်

“(ဂ) စာမျက်နှာ ၂၇ နကိုမူလနေရာကို ပြန်လာ” မှာ ပါဝင်မယ့် အလုပ်တွေကတော့

- အနောက်ဘက်ပြန်လှည့်
- နံရံဆီသွား
- ညာလှည့်
- ရွှေ့
- ညာလှည့်

ဖော်ပြခဲ့သလိုမျိုး အလုပ်တစ်ခုကို သူ့ထက် တဆင့်ပိုရိုးရှင်းတဲ့ အလုပ်တွေ(subtasks/subproblems) အဖြစ် ခွဲထုတ်မယ်၊ ရရှိလာတဲ့ အလုပ်တွေကို နောက်တဆင့် ပို၍ပို၍ ရိုးရှင်းတဲ့ အလုပ်တွေဖြစ်အောင် ထပ်ခွဲထုတ်မယ်။ အခုလိုမျိုး အလုပ်တစ်ခုကို တဆင့်ပြီးတဆင့် ရိုးရှင်းသည်ထက် ရိုးရှင်းပြီး ဖြေရှင်းရ လွယ်ကူသည်ထက် လွယ်ကူလာတဲ့ အလုပ်တွေခွဲထုတ်တဲ့ လုပ်ငန်းစဉ်ကို problem decomposition လို့ခေါ်တယ်။

၄.၂ Decomposition

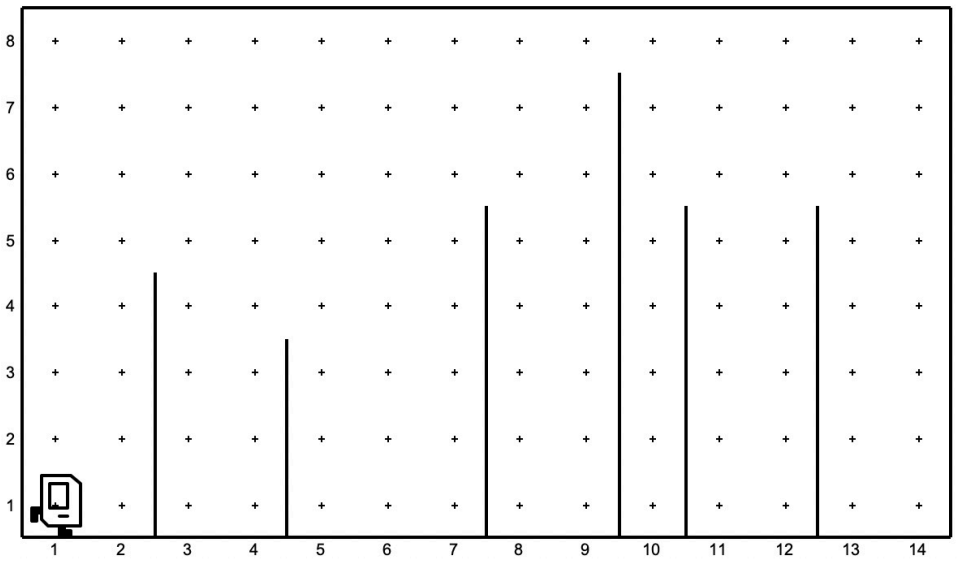
၄.၃ Top-Down Approach and Problem Decomposition

ကားရဲလ်ကို ၄.၂ တန်းရှိရင် ကျော်ရမယ်၊ မရှိရင် ရှေ့ကို ရွှေ့။ ဒါကို ၁၃ ခါကျော့ခိုင်း ရုံပါပဲ။ အလားတူ အခြား ကမ္ဘာတစ်ခု အတွက်လည်း အလုပ်လုပ်အောင် ရေးပေးရမှာပါ။ ရိုပ်သာလမ်းအရေအတွက်က ၁၄ လမ်း ပုံသေဖြစ်မယ်။ လမ်းအရေအတွက်က ကမ္ဘာတစ်ခုနဲ့တစ်ခု တူချင်မှ တူမယ်။ တန်းတွေရဲ့ အမြင့်နဲ့ နေရာတွေလည်း မတူဘူးလို့ ယူဆပါ။

ပရိုဂရမ်စမရေးခင် ဘယ်လိုခိုင်းရမလဲ စိတ်ကူးကြည့်လျှင် ကားရဲလ်ကို တန်းရှိရင် ကျော်ခိုင်း၊ မရှိရင် ရှေ့ကို ရွှေ့ခိုင်း။ ဒါကို ၁၃ ခါကျော့ခိုင်း ရုံပါပဲ။ တန်းအောက်ခြေကနေ အခြားဘက်ကို ဘယ်လိုသွားမလဲ

(က) စာမျက်နှာ ၂၇

ပုံ ၄.၂ – ကားရဲလ်နှင့် ကားရဲလ်၏ကမ္ဘာ



၄.၄ Precondition and Postcondition