DUM č. 14 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 30.06.2014

Ročník: studenti semináře

Anotace DUMu: Shrnutí druhého projektu - simulace zajíců v C/Allegro, rady k obvyklým chybám, možná vylepšení, kompletní cenzurovaný (kvůli zabránění opisovat) kód ke kontrole hotových úkolů.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
2007-13 OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt: Simulace života zajíce v C/Allegro
Shrnutí

Úvod

Projekt simulátoru lze po předchozí lekci považovat za hotový. Nabízí se samozřejmě řada doplnění a vylepšení. Implementace v C a Javě (Greenfootu) si nejsou zcela podobné, což je dáno tím, že Greenfoot nabídl již velkou část aplikace naprogramovanou v podobě běhového prostředí, což velmi zjednodušilo práci. V projektu v Greenfootu jsme se zabývali de facto jen vysokoúrovňovým programováním konkrétních funkcionalit a chování objektů, což byla spíše kodérská než analytická práce. Náročnější otázky jako uchování a procházení seznamu hráčů jsme vůbec neřešili, protože jsme používali již hotové a optimalizované objekty a metody běhového prostředí. V jazyce C s knihovnou Allegro jsme sice měli k dispozici užitečné funkce pro práci s obrázky, klávesnicí a myší, ale i tak jsme programovali poměrně nízkoúrovňově.

Většina práce na projektu v C spočívala v intelektuálně i programátorsky složité implementaci zřetězeného seznamu, vyhledávání, vkládání a komplikované alokaci paměti.

Lze přiznat, že takovýto projekt není úplně typickým vhodným příkladem pro jazyk C a minimálně by bylo výhodnější použít alespoň C++ s jeho objektovou nadstavbou. Přesto se v jazyce C programují i rozsáhlé aplikace a příkladem ze sféry open source může být grafický program GIMP, jehož zdrojové kódy v jazyce C lze volně stáhnout ze stránky http://www.gimp.org/downloads/ a upravit podle zájmu (nadaným studentům např. v maturitním semináři je dobré ukázat jak takový projekt zkompilovat s nějakou drobnou úpravou např. v uspořádání menu).

V této lekci provedeme shrnutí celého zdrojového kódu. Studenti by měli doplnit kompletní dokumentaci, pokud tak neučinili už během programování. K dokumentaci kódu v jazyce C (a dalších) je vhodné používat dokumentační nástroj Doxygen (http://www.stack.nl/~dimitri/doxygen/), který je syntakticky podobný JavaDocu, takže se studenti nemusí učit novou formu zápisu. Tento program je ale potřeba spustit samostatně stejně jako kompilátor, aby vygeneroval dokumentaci z komentářů ve zdrojovém kódu. Výstup je v HTML, CHM, PDF a dalších formátech.

Shrnutí obvyklých chyb a problémů

- Dbejte na to, abyste všechny funkce knihovny Allegro a také její direktivy (SCREEN_W, SCREEN_H) nebo globální proměnné (screen, key, apod.) používali až po volání inicialiace knihovny (v našem kódu probíhá ve funkci init). Jinak se program chová buď nepředvídatelně nebo padá zcela náhodně.
- Vždy u každé alokace paměti (včetně načítání obrázků funkcemi Allegra apod.) testujte, zda se podařilo paměť alokovat (načíst obrázek).
- Před použitím ukazatele testujte, zda není NULL. Obvyklý důvod pádu aplikace s chybou „Segmentation fail“ nebo „Shutting down Allegro due to signal #11“ je špatně alokovaná paměť nebo předávání ukazatele s již dealokovanou pamětí.

- Dávejte si pozor na správný formát obrázků BMP pro načítání funkcemi Allegra. Knihovna Allegro vyžaduje 24bitové obrázky s nerozšířenou hlavičkou. Tyto obrázky získáte např. konverzí v programu IrfanView nebo exportem z GIMPu, za předpokladu, že nastavíte při exportu: Volby kompatibility, zaškrtnout „Nezapisovat informace o barevném prostoru“; Pokročilé volby: 24 bitů (R8 G8 B8).
- Části obrázku, které mají být průhledné, mají být v Allegru sytě růžovou barvou, je to barva s kódem (255,0,255), tedy plné kanály červené a modré, žádná zelená. Allegro takto podporuje tzv. falešnou průhlednost, takže nepoužívejte u obrázků antialiasing s barvou pozadí (hladké barevné přechody), protože za průhlednou bude považován jen tento sytý odstín a ostatní odstíny zůstanou nezprůhledněné.
- Snažte se kontrolovat a testovat vše, co napíšete, jakmile je to možné, i za cenu zdržení psaním dalšího kódu. Chyby, které se objeví později ve starém kódě se hůř hledají. Představujte si všechny možné „rizikové“ situace. Například při vkládání do řetězeného seznamu počítejte nejen s případem, kdy vkládáte mezi dva existující prvky, ale i s obskurními případy, jako je jednoprvkový seznam (vkládáte před nebo za jediný prvek seznamu) nebo prázdný seznam.
- Pokud se program někde chová divně, používeje pomocné výpisy. Lze k tomu použít funkci fprintf z knihovny stdio.h a vypisovat do chybového výstupu, například takto:

```
#include <stdio.h>
fprintf(stderr,"chyba, pocet hracu: %i",getActorCount());
```

- Další alternativou k výpisům chyb je použití funkce `allegro_message("text")`.
- Hlídejte v podmínkách správné používání operátorů `&&` a `||` a jejich kombinace závorkujte.
- Nezapomeňte dealokovat veškerou paměť, kterou jste alokovali, včetně načtených obrázků.
- Pro vykreslování používejte pomocný buffer, který pak jednou funkcí přepíšete přes screen. Předejdete tak problikávání obrazovky.
- Zkontrolujte, že všechny funkce a proměnné volané uvnitř časovače jsou při inicializaci uzamčené LOCK direktivami.

Rozšíření a nedostatky

Námi vytvořené řešení má potenciál k řadě rozšíření za úkol nebo jako dodatečné práce či jako závěrečného úkolu. Obsahuje také několik nedostatků v rámci zjednodušení.

- Hráči se vykreslují postupně popořadě podle svého pořadí. To způsobuje, že zajíc nebo liška mohou být vykresleni pod mrkví. Bylo by lepší nejprve vykreslit mrkve, potom zajíce pak lišky. Aby se předešlo zdržování kreslící smyčky trojím procházením celého seznamu, bylo by vhodné vytvořit další tři seznamy, které by obsahovaly jen lišky, jen zajíce a jen mrkev. Tyto seznamy by nemusely být už seřazené.
- Bylo by vhodné v časovači (přerušení) testovat jak dlouho už probíhá vykreslování (tj. že časovač byl zavolán znovu aniž by bylo dokončeno vykreslení. To může být způsobeno

velkým množstvím rozmnožených zajíců, kteří přibývají geometrickou řadou. V takovém případě by se simulace zastavila, aby nedocházelo k zablokování kurzoru myši a testů klávesnice.

- Bylo by dobré omezit pohyb hráčů na menší herní plán, aby se nepohybovali pod tlačítky a zamezit vkládání hráče na místo tlačítek.
- Vhodné rozšíření by bylo vytvořit posuvník s nastavením rychlosti simulace. Tlačítko s možností aktivace popisků hráčů (pořadové číslo a souřadnice s rychlostí a směrem).
- Lišky by se také měly množit, je vhodné implementovat bonusy: žraní mrkve, vyhladovění.
- Scénu by mělo jít ukládat a načítat do souboru.
- Vypisování nápovědy a statistik.

Kompletní cenzurovaný kód

```
#□ <allegro.h>
#□ <math.h>
#□ <time.h>

#□ AT_FOX     0
#□ AT_RABBIT  1
#□ AT_FRABBIT 2
#□ AT_CARROT  3

#□ CHECK_TYPE(t) (t□AT_FOX □ t□AT_RABBIT □ t□AT_FRABBIT □ t□AT_CARROT)

#□ KEY_DELAY □
#□ MOUSE_DELAY □
#□ CARROT_EATABLE_TIME 200
#□ CARROT_ROTTING_TIME 600

#□ ICON_SIZE 30

□ sActor {
   □ type;
   □ □ x;
   □ □ y;
   □ □ speed;
   □ □ direction;
   □ sActor □ next;
   □ sActor □ prev;

   □ □ count;
   □ □ maxCount;
   □ □ liveTime;
};
```

```
□ □ □ firstActor      = NULL;
□ □   actorCount      = 0;
  □   addActor        = -1;
volatile □  simulationPause = □;
volatile □  close_by_button = □;
volatile □  timeCounter     = 0;
volatile □  keyDelay        = □;
volatile □  mouseDelay      = 0;
volatile □  doActivity      = 0;
volatile □  timeDivider     = 1;


void close_button_handler(void) {
   close_by_button = □;
}
END_OF_FUNCTION(close_button_handler)

void timeInterrupt() {
   if (!□) {
      timeCounter□;
      timeCounter□=1000;
      if(□(timeCounter□timeDivider)) doActivity = □;
   }
   keyDelay□;
   if (keyDelay□0) keyDelay=□;
   mouseDelay□;
   if (mouseDelay□0) mouseDelay=□;
}
END_OF_FUNCTION(timeInterrupt)

int keyPressed(int k) {
   return (keyDelay□0)□(key[k]==□)□(keyDelay=□);
}

int getActorCount() {
   return actorCount;
}

□ createActor(□ type, □ □ x, □ □ y, □ □ speed, □ □ direction) {
   □ □ □ actor = (struct sActor □) malloc(sizeof(□ □));
   □ □ □ it;
   if (!□ □
       !(□(type)) □
       x□SCREEN_□ □ y□SCREEN_□) {
     return NULL;
   }
   actor->type      = type;
   actor->□         = x;
   actor->□         = y;
```

```
   actor->speed     = speed;
   actor->direction = direction;

   actorCount□;

   if (□ == □) {
      actor->next = □;
      actor->prev = □;
      firstActor  = □;
      return actor;
   }

   //search biggest smaller
   for(it=firstActor; (it->□<actor->□ □
                        (it->□□actor->□ □ it->□ □ actor->□))□
                       it->next□NULL; it = it->□);
   if(it->next!=□ □
      !(it->□<actor->□ □ (it->□□actor->□ □ it->□ □ actor->□))) {
      it = it->□;
   }

   actor->□ = it;
   if (□ == □) {
      actor->next      = □;
      firstActor->prev = □;
      firstActor       = □;
   } else {
      actor->next       = □;
      it->next          = □;
      if(actor->□!=NULL) {
         actor->□->□ = actor;
      }
   }

   return actor;
}

void removeActor(□ actor) {
   if (!□) return;
   if (actor->□ == NULL) {
      firstActor = actor->□;
      if (actor->□ != □) {
         actor->□->□ = NULL;
      }
   } else {
      actor->□->□ = actor->□;
      if (actor->□!=□) {
         actor->□->□ = actor->□;
      }
```

```
   }

   actorCount□;

   free(actor);
}
void deallocActors() {
   □ it;
   actorCount=□;
   if (firstActor==□) return;
   if (firstActor->□==□) {
      free(firstActor);
      firstActor = □;
      return;
   }
   for (it=firstActor->□;it!=□ □ it->□!=□;it = it->□) {
      free(it->□);
   }
   free(□);
   firstActor = □;
}

void moveActor(□ actor) {
   □ it;
   int dx = (□) □(actor->□*□((actor->□)/□.0*□));
   int dy = (□) □(actor->□*□((actor->□)/□.0*□));
   if (actor->□+□>=0 □ actor->□+□<SCREEN_□ □
       actor->□+□>=0 □ actor->□+□<SCREEN_□) {
       actor->□+=□;
       actor->□+=□;
       if (actor->□!=□) {
          //search smallest bigger actor (if actor is now bigger)
          for(it=actor->□; it□ NULL □ (it->□<actor->□ □
                                (it->□□actor->□ □ it->□ □ actor->□))□
                               it->□!=□; it = it->□);
          //get last smaller or equal
          if(it->□!=□ □
             !(it->□<actor->□ □ (it->□□actor->□ □ it->□ □ actor->□)))
{
            it = it->□;
          }
          if (□ != □) {
             //deconnect actor
             if(actor->□ == □) {
                firstActor = actor->□;
             } else {
                actor->□->□ = actor->□;
             }
             if (actor->□ != □) {
```

```
                actor->□->□ = actor->□;
             }

             //reconnect actor
             actor->prev = □;
             actor->next = □;
             it->next    = □;
             if (actor->□!=□) {
                actor->□->□ = □;
             }
          }
       }
       if (actor->□!=□) {
          //search biggest smaller or equal actor
          //(if actor is now smaller)
          for(it=actor->□; it!=□ □ (it->□>actor->□ □
                                (it->□□actor->□ □ it->□ □ actor->□))□
                               it->□!=□; it = it->□);
          //get last bigger
          if(it->□!=□ □
             !(it->□>actor->□ □ (it->□□actor->□ □ it->□ □ actor->□)))
{
            it = it->□;
          }
          if (□ != □) {
             //deconnect actor
             actor->□->□ = actor->□;
             if (actor->□ !=□) {
                actor->□->□ = actor->□;
             }

             //reconnect actor
             actor->prev = □;
             actor->next = □;
             it->prev    = □;
             if (actor->□!=□) {
                actor->□->□ = □;
             } else {
                firstActor = □;
             }
          }
       }
   }
}

void moveAllActors() {
   □ it = firstActor;
   for (;it!=□;it = it->□) {
      moveActor(□);
```

```
   }

}

int loadActorImages(□ images) {
   □ pal;
   images[AT_FOX]=□("fox.bmp",pal);
   if(!images[AT_FOX]) {
      allegro_message("Cannot found fox.bmp.");
      return 0;
   }
   images[AT_RABBIT]=□("rabbit.bmp",pal);
   if(!images[AT_RABBIT]) {
      allegro_message("Cannot found rabbit.bmp.");
      return 0;
   }
   images[AT_FRABBIT]=□("frabbit.bmp",pal);
   if(!images[AT_RABBIT]) {
      allegro_message("Cannot found frabbit.bmp.");
      return 0;
   }
   images[AT_CARROT]=□("carrot.bmp",pal);
   if(!images[AT_CARROT]) {
      allegro_message("Cannot found carrot.bmp.");
      return 0;
   }
   return 1;
}

void drawActor(□ actor, □ buffer, □ images) {

   if (!□ □ !□ □ !□) return;
   switch(actor->□) {
      case AT_FOX:
      case AT_RABBIT:
      case AT_FRABBIT:
         if (actor->direction□□ □ actor->direction□□) {
            □(buffer, images[□], actor->□-images[□]->□/□,
              actor->□-images[□]->□/□, □(actor->direction/□.0*□));
         } else {
            □(buffer, images[□], actor->□-images[□]->□/□,
              actor->□-images[□]->□/□, □(actor->direction/□.0*□));
         }
         break;
      case AT_CARROT:
         if (actor->□<CARROT_EATABLE_TIME) {
            □(buffer, actor->□-images[□]->□/□+□, actor->□-□->□/□+□,
              actor->□+□->□/□-□, actor->□+□->□/□-□, □(32,106,35));
         }
```

```
         if (actor->□□CARROT_EATABLE_TIME □
             actor->□□CARROT_ROTTING_TIME) {
            □(buffer, actor->□-□->□/□+□, actor->□-□->□/□+□,
              actor->□+□->□/□-□, actor->□+□->□/□-□, □(249,117,0));
         }
         if (actor->□>CARROT_ROTTING_TIME) {
            □(buffer, actor->□-□->□/□+□, actor->□-□->□/□+□,
              actor->□+□->□/□-□, actor->□+□->□/□-□, □(94,73,34));
         }
         □(buffer, □, actor->□-□->□/□, actor->□-□->□/□);
         break;
      default:
         break;
   }
}

void drawAllActors(□ buffer, □ images, □ info) {
   □ it = firstActor;
   int i = 0;
   for (;it!=□;it = it->□) {
      i□;
      drawActor(□, buffer, images);
      if (info) {
         □(buffer,font, it->□, it->□, □(255,255,255), -1,
           "%i [%i,%i]", i, it->□, it->□);
         □(buffer,font, it->□, it->□+12, □(255,255,255), -1,
           "S%i D%i", it->speed, it->direction);
      }
   }
}

void init() {
    int depth, res;
    allegro_init();
    depth = desktop_color_depth();
    if (depth == 0) depth = 32;
    set_color_depth(depth);
    res = set_gfx_mode(□, □, □, 0, 0);
    if (res != 0) {
         exit(-1);
    }

    install_timer();
    install_keyboard();
    install_mouse();
    □_cursor();
    □(time(□));

    LOCK_□(timeCounter);
```

```
    LOCK_□(timeDivider);
    LOCK_□(simulationPause);
    LOCK_□(keyDelay);
    LOCK_□(mouseDelay);
    LOCK_□(doActivity);
    LOCK_□(close_by_button);
    LOCK_FUNCTION(□);
    LOCK_FUNCTION(□);

    set_□(close_button_handler);
}

void deinit() {
    clear_keybuf();
}

int loadBackground(□ bgBuffer) {
   int x, y;
   □ palette;
   □ image = □("grass.bmp",palette);
   if (!□) return 0;

   □bgBuffer = □(SCREEN_□,SCREEN_□);
   if (!□bgBuffer) return 0;

   for(y=0;y<SCREEN_□/image->□;y□) {
     for(x=0;x<SCREEN_□/image->□;x□) {
        □(image, □bgBuffer, 0, 0, □*image->□, □*image->□, □, □);
     }
   }
   return 1;
}

int getDistance(□ a1, □ a2) {
   return □((□-□)*(□-□)+(□-□)*(□-□));
}

int interact(□ actor) {
    □ it;
    int random=0;
    int distance=0;

    if (□==□){
       return 0;
    }

    if (actor->□ != □) {
       for (it=actor->□;it!=□ □ □+60>=□; it = it->□) {
          distance = getDistance(□, □);
```

```
            random = □()□10;
            if (actor->type□AT_FRABBIT □ it->type□AT_RABBIT □
                random □□ □ distance □15) {
               createActor((□?AT_RABBIT:AT_FRABBIT),actor->□,
                           actor->□,2,0);
               return □;
            }
            if (actor->type==AT_FOX □ (it->type□AT_RABBIT □
                it->type□AT_FRABBIT) □ distance □60) {
               it = it->□;
               removeActor(□);
            }
         }
      }

      if (actor->□ != □) {
         for (it=actor->□;it!=□ □ □-60<=□; it = it->□) {
            distance = getDistance(□, □);
            random = □()□10;
            if (actor->type□AT_FRABBIT □ it->type□AT_RABBIT □
                random □□ □ distance □15) {
               createActor((□?AT_RABBIT:AT_FRABBIT),actor->□,
                           actor->□,2,0);
               return □;
            }
            if (actor->type==AT_FOX □ (it->type□AT_RABBIT □
                it->type□AT_FRABBIT) □ distance □60) {
               it = it->□;
               removeActor(□);
            }
         }
      }
      return 0;
}

void interactAll() {
   □ it = firstActor;
   for (;it!=□;it = it->□) {
      if (interact(□) □ it->□!=□) it=it->□;
   }
}

void oneLapActivity() {
   □ actor = firstActor;
   for (;actor!=□;actor = actor->□) {
      switch(actor->□) {
         case AT_FOX:
            if (□()□6==0) {
               actor->direction□=□()□41-20;
```

```
            }
            actor->speed□=□()□5-2;

            actor->direction□=360; actor->direction□=360;
            actor->speed = (actor->speed<0?0:(actor->speed>10?10:ac-
tor->speed));
            break;
         case AT_FRABBIT:
         case AT_RABBIT:
            actor->direction+=(rand()%7)*15-45;
            actor->speed=rand()%11+4;

            actor->direction+=360; actor->direction%=360;
            actor->speed = (actor->speed□0?0:(actor->speed□10?10:□));
            break;
         case AT_CARROT:
            actor->liveTime□;
            actor->liveTime□=800;
            break;
         default:
            break;
      }
   }
   interactAll();
}

void drawIcons(□ buffer, □ images) {
   int x = SCREEN_□-□*□/□;
   int y = □/□;
   int i = 0;
   double scale;

   for (i = 0; i<6; i□, y□=□/□.0*□) {
      □(buffer,x,y,x+□,y+□,□(255,150,0));
      □(buffer,x+□,y+□,x+□-□,y+□-□,□(255,255,150));

      switch(i) {
         case 0:
            if(□) {
               □(buffer,x+□,y+□,x+□,y+□-□,x+□-6,y+□/□,□(0,150,0));
            } else {
               □(buffer, x+□/□.0,y+□,x+□*□/□.0,y+□-□,□(0,150,0));
               □(buffer, x+□*□/□.0,y+□,x+□*□/□.0,y+□-□,□(0,150,0));
            }
            break;
         case 1:
            line(buffer,x+□,y+□,x+□-□,y+□-□,□(150,0,0));
            line(buffer,x+□,y+□,x+□-□,y+□-□,□(150,0,0));
            line(buffer,x+□,y+□,x+□-□,y+□-□,□(150,0,0));
```

```
            line(buffer,x+□,y+□-□,x+□-□,y+□,□(150,0,0));
            line(buffer,x+□,y+□-□,x+□-□,y+□,□(150,0,0));
            line(buffer,x+□,y+□-□,x+□-□,y+□,□(150,0,0));
            break;
         case 2:
             scale = □/□;
             □(buffer, images[AT_RABBIT], x+□, y+(□+□-((□-6)*□))/□.0,
               □-□, (□-□)*□);
             break;
         case 3:
             □ = □/□;
             □(buffer, images[AT_FRABBIT], x+□, y+(□+□-((□-□)*□))/□.0,
               □-□, (□-□)*□);
            break;
         case 4:
             □ = □/□;
             □(buffer, images[AT_FOX], x+□, y+(□+□-((□-□)*□))/□.0,
               □-□, (□-□)*□);
            break;
         case 5:
             □ = □/□;
             □(buffer, images[AT_CARROT], x+□, y+(□-((□-□)*□))/□.0,
               □-□, (□-□)*□);
            break;
      }
   }

}

void checkMouse(□ image) {
   int x = SCREEN_□-□*□/□;
   int y = □/□;
   int i = 0;

   if(□>0) return;

   if (mouse_□ □ □) {

      if(□(addActor)) {
         createActor(□,mouse_□+□->□/□,mouse_□+□->□/□, (□!=□?2:0),0);
         addActor=□;
         set_□(NULL);
      }

      for (i = 0; i<6; i□, y□=□/□.0*□) {
         if (mouse_x□x □ mouse_x□x+□ □
             mouse_y□y □ mouse_y□y+□) {

             mouseDelay=□;
```

```
            switch(i) {
               case 0:
                  simulationPause□simulationPause;
                  break;
               case 1:
                  simulationPause=□;
                  □(timeInterrupt);
                  deallocActors();
                  □(timeInterrupt,10);
                  break;
               case 2:
                  □(image[AT_RABBIT]);
                  addActor = AT_RABBIT;
                  break;
               case 3:
                  □(image[AT_FRABBIT]);
                  addActor = AT_FRABBIT;
                  break;
               case 4:
                  □(image[AT_FOX]);
                  addActor = AT_FOX;
                  break;
               case 5:
                  □(image[AT_CARROT]);
                  addActor = AT_CARROT;
                  break;
            }
         }
      }
   }
}


int main() {
    init();

    □ bgBuffer;
    □ actorImages[3];
    □ buffer = □(SCREEN_□,SCREEN_□);
    if (!□) return 1;

    □ lastActor = NULL;

    if (!loadBackground(□bgBuffer)) return 2;
    if (!loadActorImages(actorImages)) return 3;

    install_int(timeInterrupt,10);
```

```
while (!key[KEY_ESC] □ !key[□] □ !close_□) {
   □(bgBuffer, buffer, 0, 0, 0, 0, SCREEN_□, SCREEN_□);
   drawAllActors(□, □, 0);
   drawIcons(□,□);
   show_mouse(□);
   □(buffer, screen, 0, 0, 0, 0, SCREEN_□, SCREEN_□);

   checkMouse(□);

   if (keyPressed(KEY_F)) lastActor->type=AT_FOX;
   if (keyPressed(KEY_T)) {
      doActivity=1;
   }
   if (keyPressed(KEY_S)) {
      simulationPause□simulationPause;
   }
   if (keyPressed(KEY_R)) lastActor->type=AT_RABBIT;
   if (keyPressed(KEY_A)) lastActor->type=AT_FRABBIT;
   if (keyPressed(KEY_C)) {
      lastActor->type=AT_CARROT;
      lastActor->speed=0;
      lastActor->count=0;
      lastActor->maxCount=10;
      lastActor->liveTime=0;
   }

   if (keyPressed(KEY_PLUS_PAD)) timeDivider□;
   if (keyPressed(KEY_MINUS_PAD) □ timeDivider>□) timeDivider□;
   if (keyPressed(KEY_SLASH_PAD) □ lastActor->□!=NULL) {
      lastActor = lastActor->prev;
   }
   if (keyPressed(KEY_ASTERISK) □ lastActor->□!=NULL) {
      lastActor = lastActor->next;
   }
   if (keyPressed(KEY_K)) {
      removeActor(lastActor);
      lastActor=□;
   }
   if (keyPressed(KEY_O)) {
      lastActor->direction□=10;
      lastActor->direction□=360;
   }
   if (keyPressed(KEY_N)) {
      lastActor = createActor(AT_RABBIT,SCREEN_□/2,
                              SCREEN_□/2,10,0);
   }

   if(□) {
      oneLapActivity();
```

```
            moveAllActors();
            doActivity=□;
        }
    }

    □_int(timeInterrupt);

    deinit();
    destroy_bitmap(□);
    deallocActors();
    return 0;
}
END_OF_MAIN()
```