DUM č. 13 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 30.06.2014

Ročník: studenti semináře

Anotace DUMu: Interakce hráčů, implementujeme funkci interact pro detekci jiného hráče v okolí a v případě, že jsou splněny podmínky vzdálenosti a typu hráče, dojde k pozření zajíce liškou, nebo zplození nového zajíce zajícem a zaječkou.

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

Projekt: Simulace života zajíce v C/Allegro
Interakce hráčů

Úvod

Projekt je již téměř hotový, ale pro plnohodnotnou simulaci zatím chybí její podstatná část a to je interakce objektů. Má-li být simulace užitečná měli bychom být schopni implementovat množení zajíců, jejich žraní liškou a možnost nakrmit se mrkví a také vyhladovět.

Tyto činnosti jsme snadno implementovali v Greenfootu díky poděděné metodě ze třídy Actor, která nám vracela kráče v daném okolí. Tuto metodu ale nemáme nyní k dispozici, ani žádnou podobnou funkci.

Implementace v naší hře bude spočívat v hledání v řetězeném seznamu. Díky tomu, že jsme implementovali jeho řazenou variantu, bude vyhledávání rychlejší, jelikož se nebude muset projít sekvenčně celý seznam, ale bude stačit projít jen okolí po nejbližšího předchůdce a následovníka, který podmínku vzdálenosti nesplňuje. Bohužel ale herní plocha není lineární a především naše řazení není lineární. Proto budeme muset hledat ne po prvního, jehož vzdálenost je delší než požadovaná, ale po prvního, jehož vzdálenost je delší než požadovaná na ose x. To v případě, že máme variantu řazení podle osy x a dále podle y. V případě „řazení kružnicí“, tedy podle vzdálenosti od počátku budeme hledat po hráče jehož vzdálenost v rozdílu s naší je už větší než požadovaná, ale stejně musíme testovat každého z hráčů, zda vzdálenost opravdu odpovídá, protože na kružnici s větším poloměrem mohou být hráči od sebe navzájem velmi vzdálení, byť jejich poloměr k počátku je stejný. Neřazená varianta seznamu vyžaduje prohledat celou lineárně, což bude u většího počtu hráčů velmi výpočetně náročné.

Úkoly s řešeními

1. Implementujte funkci getDistance, která bude vracet celočíselnou vzdálenost dvou hráčů předaných ukazatelem v parametrech funkce.

   Řešení je jednoduchá matematika, použijeme odmocninu z matematické knihovny math.h, kterou už máme připojenou, pro výpočet mocniny použijeme raději rychlejší násobení (z definice mocniny), než funkci pro mocnění:

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

2. Napište funkci interact(), která bude přebírat adresu jednoho hráče a hledat v okolí 15 pixelů hráče, se kterými bude interagovat. Prozatím zkusíme jen množení zajíců. Použijte generátor (pseudo)náhodných čísel, abyste zajistili, že poměr zplozených potomků zajíc:zaječka:nic bude 1:1:8. Z důvodů, které poznáme později (jde o „řetězovou reakci“ s interakcí s právě narozeným zajícem) bude důležité předávat informaci o tom, že byl zplozen nový zajíc. Vhodné je proto vracet int a to tak, že 0 je obvyklá návratová hodnota a 1 je při zplození nového zajíce.

   K detekci použijeme cyklus, pomocí kterého budeme procházet nejprve předchůdce a potom následovníky předaného hráče. Procházet budeme tak dlouho, dokud se souřadnice x (pro variantu s řazením podle x a y) bude lišit nejvýše o 15 pixelů. U každého hráče ale přepočítáme jeho vzdálenost, protože je možné, že je hodně vzdálený na ose y. Studenti s implementací neřazeného seznamu prochází seznam celý až po hráče, který je na okraji. Jistě každého studenta napadne, že správně bude implementovat test na překrývání zaječky s zajícem pouze u jednoho pohlaví, aby nedocházelo k dvojitému množení v jednom hracím kole. Z logiky věci budeme implementovat test pouze u zaječky, v případě úspěchu vytvoříme nového zajíce na stejném místě. Opět další logická úvaha nás zavede k tomu, že s testem končíme ihned, jakmile najdeme v okolí prvního zajíce – tj. neumožníme plodit v jednom kole více než jedenkrát. Řešení poskytneme až v dalším úkolu.

3. Do funkce interact doplňte žraní zajíce liškou, proto pokud je hráč liška a v jeho okolí 60 pixelů se nachází zajíc (či zaječka), nechte jej sežrat (dealokovat).

   Do funkce se doplní pouze další podmínka. Úkol v sobě skrývá jednu záludnost, která bude způsobovat nepozornému studentovi pády aplikace kvůli použití špatného ukazatele. Důvod je ten, že pokud v cyklu odstraníme hráče na kterého jsme měli zrovna ukazatel, stane se tento ukazatel neplatným a nejsme schopni pokračovat v prohledávání seznamu. Před dealokací hráče je nutné nejprve správně upravit ukazatel, kterým seznam procházíme. Ale opět pozor! Pokud nastavíme ukazatel na následujícího hráče, tak musíme tělo cyklu zopakovat s ním, protože jinak se tělo dokončí a přejde se na jeho následovníka, zatímco on bude „ušetřen“. Proto je vhodnější nastavit ukazatel zpět na předchůdce…

   ```
   int interact(□ actor) {
       struct sActor * it;
       int random=0;
       int distance=0;

       if (actor==□){
   ```

```
        return 0;
    }

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

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

4. Napište funkci interactAll, která bude volat funkci interact nad každým z hráčů. Doplňte ji na vhodné místo programu.

   Funkce vypadá podobně jako ostatní funkce, které procházejí seznam a například vykreslují hráče. Vhodné umístění je ve funkci oneLapActivity.

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

5. BONUS: Implementujte žraní mrkve, výpočet energie (s každým krokem úbytek, s nažráním se přibude dané množství), smrt vyhladověním…

   Jelikož se jedná o implementaci téhož, co jsme implementovali už v Greenfootu, je úkol vhodný jako „písemkový“ či „zkouškový“. je třeba si dávat pozor jen na odlišnosti jazyka. Místo atributů objektu implementujeme nové položky do struktury sActor. Úbytek energie patří do funkce oneLapActivity, stejně jako volání funkce interact. Řešení vzhledem k použití úkolu při zkoušení neposkytuji.
6. BONUS: Spočítejte pomocí funkce time() čas strávený ve vykreslovacím cyklu – stačí jen délku provádění oneLapActivity. Vypisujte průměr za posledních 10 spuštění této funkce. Změřte, jak velký je rozdíl při 1, 10, 30, 50, 100 a 1000 hráčích pro jednotlivé typy implementací (seznam bez řazení, řazení pořadím na ose x a řazení do kružnic). Během měření zakomentujte ve funkci interact řádek se žraním a plozením zajíců nebo vložte do scény jen zajíce (žádné zaječky ani lišky). Je rozumné si pro větší čísla nechat hráče v požadovaném počtu vygenerovat for cyklem.

Úkoly

1. Implementujte funkci getDistance, která bude vracet celočíselnou vzdálenost dvou hráčů předaných ukazatelem v parametrech funkce.
2. Napište funkci interact(), která bude přebírat adresu jednoho hráče a hledat v okolí 15 pixelů hráče, se kterými bude interagovat. Prozatím zkusíme jen množení zajíců. Použijte generátor (pseudo)náhodných čísel, abyste zajistili, že poměr zplozených potomků zajíc:zaječka:nic bude 1:1:8. Z důvodů, které poznáme později (jde o „řetězovou reakci“ s interakcí s právě narozeným zajícem) bude důležité předávat informaci o tom, že byl zplozen nový zajíc. Vhodné je proto vracet int a to tak, že 0 je obvyklá návratová hodnota a 1 je při zplození nového zajíce.
3. Do funkce interact doplňte žraní zajíce liškou, proto pokud je hráč liška a v jeho okolí 60 pixelů se nachází zajíc (či zaječka), nechte jej sežrat (dealokovat).
4. Napište funkci interactAll, která bude volat funkci interact nad každým z hráčů. Doplňte ji na vhodné místo programu.
5. BONUS: Implementujte žraní mrkve, výpočet energie (s každým krokem úbytek, s nažráním se přibude dané množství), smrt vyhladověním…
6. BONUS: Spočítejte pomocí funkce time() čas strávený ve vykreslovacím cyklu – stačí jen délku provádění oneLapActivity. Vypisujte průměr za posledních 10 spuštění této funkce. Změřte, jak velký je rozdíl při 1, 10, 30, 50, 100 a 1000 hráčích pro jednotlivé typy implementací (seznam bez řazení, řazení pořadím na ose x a řazení do kružnic). Během měření zakomentujte ve funkci interact řádek se žraním a plozením zajíců nebo vložte do scény jen zajíce (žádné zaječky ani lišky). Je rozumné si pro větší čísla nechat hráče v požadovaném počtu vygenerovat for cyklem.