Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów 2

Bomberman:
gra komputerowa

autor	Hubert Przegendza
prowadzący	dr inż. Adam Gudyś

rok akademicki	2017/2018
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	czwartek P, 10:00 – 11:30
grupa	2
sekcja	2
termin oddania sprawozdania	2018-06-20
data oddania sprawozdania	2018-06-20

1 Treść zadania

Napisać grę typu Bomberman.

2 Analiza, projektowanie

Przed przystąpieniem do pisania kodu należy zrozumieć co znaczy ”gra typu Bomberman”. Jest to gra, w której gracz porusza się w czterech kierunkach (góra, dół, lewo, prawo) po dwuwymiarowej prostokątnej mapie. Na tej mapie znajdują się przeszkody, przez które gracz przejść nie może - pudła i filary. Gracz może postawić w miejscu na mapie, w którym aktualnie się znajduje, bombę, która po pewnym czasie eksploduje, przez co na mapie powstanie krzyż ognia ze środkiem w miejscu, w którym bomba się znajdowała. Wielkość tego krzyża zależy od posiadanych przez gracza ulepszeń.

Ulepszenia znajdują się w niektórych pudełkach i ujawniane są po jego zniszczeniu. Pudło ulega zniszczeniu, kiedy znajdzie się w zasięgu eksplozji bomby. Jeżeli gracz znajdzie się w zasięgu eksplozji bomby, to zostaje zraniony, przez co może przegrać grę. Jeżeli jednak gracz posiada ulepszenie tarczy, to może przetrwać taką eksplozję kosztem utracenia po chwili tego ulepszenia.

Poza ulepszeniami zasięgu bomby oraz tarczy, w tym projekcie zawarte są jeszcze ulepszenie zwiększenia ilości możliwych do postawienia bomb oraz ulepszenie dające możliwość graczowi ”kopnięcia bomby”, czyli wprawieniu jej przez kolizję z graczem w ruch aż do napotkania przeszkody (np. filar, pudło, przeciwnik, inna bomba...).

Na mapie mogą znajdować się również przeciwnicy. Nie mogą oni stawiać bomb ani zbierać ulepszeń, ale jeśli gracz dotknie któregoś z nich, to zostanie zraniony, chyba że posiada on ulepszenie tarczy. Przeciwnicy różnią się od siebie swoim algorytmem poruszania się. Różne gry tego typu mają różne typy przeciwników (w tym projekcie zdecydowano skorzystać z 3 algorytmów opisanych dalej w rozdziale w podpunkcie ”Algorytmy”).

Aby wygrać grę lub jej etap, należy pokonać wszystkich przeciwników, czyli sprawić, by znaleźli się w zasięgu eksplozji bomby gracza i wejść do portalu, który pojawia się na środku mapy, kiedy nie ma juz aktywnych przeciwników.

2.1 Struktury

W projekcie większość danych jest przechowywane w strukturach statycznych, ponieważ okazały się one czystsze, wygodniejsze i szybsze od struktur dynamicznych, które wymagały wiele funkcji do ich obsługi, a korzystanie z nich wiązało się z dużą ilością pogorszających czytelność znaków potrzebnych do obsługi wskaźników. Wadą tego rozwiązania jest to, że nie można płynnie zmieniać niektórych wielkości. W tym projekcie wychodzenie poza pewien określony zakres w większości przypadków jest nieprawdopodobne lub wręcz niemożliwe, więc struktury dynamiczne tracą na przydatności.

Np. gracz nie jest w stanie sprawić, by na mapie znajdowało się więcej niż 20 bomb na raz, więc przechowywanie danych bomb w dynamicznej strukturze nie ma przewagi wobec przechowywania ich w statycznej tablicy o wielkości 20.

Dynamicznie zdecydowano się np. przechowywać dane mapy, dzięki czemu można zmienić jej rozmiar bez modyfikacji kodu (np. przez parametr wywołania w konsoli).

2.2 Algorytmy

W projekcie zdecydowano się skorzystac z trzech typów przeciwników, *blind*, *random* oraz *charger* i każdy z nich korzysta z osobnego algorytmu poruszania się.

Przeciwnik typu *blind* porusza się po linii prostej aż natrafi na przeszkodę, po czym zmienia kierunek najpierw na o kąt prosty w lewo, a jeśli nie może się poruszać w tym kierunku, to o kąt prosty w prawo. Jeśli nie może się ruszyć i w tym kierunku, to zawraca, tj. zmienia kierunek na przeciwny względem oryginalnego.

Przeciwnik typu *random* porusza się w przypadkowym kierunku, przy czym jeśli nie może się poruszyć w nowo obranym kierunku, to zmienia swój kierunek na przypadkowy. Robi tak okreloną ilość razy, po czym jeśli dalej się nie poruszył, to stoi w miejscu aż znowu będzie przesuwany.

Ostatni przeciwnik, typu *charger*, porusza się tak samo jak przeciwnik typu *blind* o ile w jego polu widzenia nie znajdzie się gracz, tzn. nie znajdzie się w tej samej kolumnie lub wierszu i nie będzie między nimi filaru (filar zasłania gracza). Jeśli w polu widzenia tego przeciwnika znajdzie się gracz, to ten przeciwnik zmienia swój kolor oraz tryb poruszania się. Teraz zacznie poruszać się z o wiele większą prędkością w kierunku, w którym ostatni raz zauważył gracza, niszcząc po drodze napotkane pudełka. Wraca do poprzedniego trybu dopiero wtedy, kiedy w jego polu widzenia nie ma gracza oraz jednocześnie napotka na przeszkodę, której nie potrafi przejść (tj. postawiona

bomba, inny przeciwnik, koniec mapy).

2.3 Specyfikacja treści

Ograniczoność specyfikacji problemu narzuca przeanalizowanie typu gry "Bomberman" i dopasowanie funkcjonalności projektu do potencjalnych oczekiwań konsumenta, np. czy użytkownik spodziewa się istnienia w grze ulepszeń, a jeśli tak to jakie lub jakich przeciwnikow się spodziewa.

3 Specyfikacja zewnętrzna — Instrukcja Obsługi

Program można uruchomić z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

`-h` lub -help spowoduje wypisanie w konsoli pomocy nt. obsługi przełączników oraz zakończenie działania programu
`-s` ustawi wielkość boku mapy gry, gdzie s to liczba nieparzysta (dla estetyki)
`-e` ilość przeciwników $e \in [1, s^2 * 0.75]$ i $e \in [1, 15]$
`-b` ilość pudełek $b \in [0, s^2 * 0.75 - e]$
`-u` ilość ulepszeń $u \in [0, b]$
`-t` czas gry - liczba naturalna
`-d` tryb debugowy $d = 0$ lub $d = 1$

Przykładowe użycie:

```
Boomber.exe -s 11 -e 4 -b 40 -t 60 -d 1
```

włączy grę z mapą o rozmiarze 11, ilością pudełek = 40, ilością przeciwników = 4, czasem gry 60sec i w trybie debugowym.

Jeżeli przełączniki zostaną wykorzystane w zły sposób, to komunikat:

```
"Zle uzycie.
Wpisz:
"[Nazwa_Programu] -h"  lub
"[Nazwa_Programu] help",
zeby otrzymac pomoc."
```

zostanie wypisany do konsoli, a wartości zostaną ustawione domyślnie (opisane w specyfikacji wewnętrznej).

Użycie przełącznika ”-h” lub ”help” spowoduje wyświetlenie w konsoli komunikatu:

```
"Wpisz "[Nazwa_Programu] -e [ilosc przeciwnikow (15 > x > 1)]
-b [ilosc pudelek (< 3/4 * (wielkosc mapy)^2) - ilosc przeciwnikow)]
-u [ilosc ulepszen (< ilosc pudelek]
-s [wielkosc mapy (>=7)]
-t [czas gry w sekundach]
-d [debug mode =0 || =1]""
```

Jeżeli podana po przełączniku wartość okaże się nie do zrealizowania (zmienne mocno zależą od siebie), to na jej miejsce zostanie ustawiona wartość domyślnie, o czym zostanie wypisana informacja w konsoli.

Użycie któregoś z przełączników więcej niż raz spowoduje wyświetlenie komunikatu

```
"przełącznik "-x" został użyty więcej niż jeden raz",
```

gdzie ”-x”, to owy przełącznik. Należy wtedy upewnić się, że przełącznik ”-x” jest użyty tylko raz.

Użycie wartości, która nie może być zrealizowana przez program, spowoduje wywietlenie komunikatu

```
">wartosc po "-x" jest poza zakresem. Uzyto wartosci domyslnej"
```

po czym, zostaną ustawione wartości domyślne.

Użycie błędnej wartości, która nie może być rozpoznana przez program jako liczba, spowoduje wywietlenie komunikatu

```
">wartosc po "-x" nie jest poprawna liczba. Uzyto domyslnej wartosci".
```

po czym, zostaną ustawione wartości domyślne. Należy wtedy upewnić się, że podano po przełączniku ”-x” poprawną liczbę.

Nie użycie żadnego przełącznika spowoduje wyświetlenie komunikatu

```
"Nie wykryto przelacznikow. Uzyto wartosci domyslnych".
```

po czym, zostaną ustawione wartości domyślne.

3.1 Wygląd okna

Po stworzeniu okna pojawi się na nim okno pauzy z wiadomością

```
''HELLO! PRESS SPACE TO START',
```

zachęcającą do rozpoczęcia gry. Aby to zrobić, należy nacisnąć klawisz spacja. Następnie w oknie gry pojawi się mapa oraz znajdująca się po prawej stronie okna tablica z danymi.

W tablicy z danymi wypisane są ilość punktów gracza pod tytułem "SCORE", największa dotychczasowa ilość punktów, zwana dalej "highscore", pod tytułem "HIGHSCORE" oraz pozostała ilość czasu pod tytułem "TIME".

Lewa część okna to tzw. mapa gry. Mogą znajdować się na niej:

1. ścieżka - ciemnoszare pola, wszystko może przez nie przejść oraz mogą po nich przesuwać się bomby,
2. pudełka - jasnoszare pola, nie można przez nie przejść, można je zniszczyć wybuchem bomby,
3. filary - białe pola, nie można przez nie przejść i nie da się ich zniszczyć,
4. przeciwnicy:
   (a) żółty kolor = przeciwnik typu *blind*,
   (b) turkusowy = *random*,
   (c) ciemno i jasno fioletowy = *charger*.
5. bomby - im bliżej do wybuchu, tym jaśniejsze i większe czerwone kwadraty, nic nie może przez nie przejść, poza graczem z ulepszeniem kopania bomb, po chwili wybuchają a w zasięgu wybuchu powstaje ogień,
6. ogień - żółte pola, wejście na nie zadaje obrażenia graczowi i przeciwnikom,
7. portal - animowany kafel na środku planszy w odcieniach niebieskiego, pojawia się po pokonaniu wszystkich przeciwników i wejście do niego oznacza wygranie planszy,
8. gracz - niebieski kwadrat, który pojawia się na pierwszym filarze z lewej z góry,
9. ulepszenia - romby o różnych kolorach:
   (a) czerwony - zebranie zwiększa ilość dostępnych bomb,

(b) żółty - zwiększa zasięg bomb,

(c) różowy - włącza możliwość kopania bomb,

(d) jasnoniebieski - daje graczowi tarczę.

3.2 Klawisze

1. Aby poruszać się po mapie należy, korzystać z klawiszy strzałek klawiatury.

2. Aby postawić bombę, należy nacisnąć klawisz spacji.

3. Aby włączyć pauzę gry, należy nacisnąć klawisz 'P'. Spowoduje to zatrzymanie gry oraz pokazanie ekranu pauzy z treścią:

   ```
   ''GAME PAUSED PRESS SPACE TO UNPAUSE
   SCORE XXXX'',
   ```

   gdzie XXXX to aktualna ilość punktów gracza. Po naciśnięciu klawisza spacji, okno pauzy zniknie, a gra zostanie kontynuowana (bez utraty czasu).

4. Aby wyłączyć grę, należy nacisnąć klawisz Escape.

3.3 Mechanizm gry

1. Punktacja: niszczenie pudełek przez gracza (10 punktów za pudełko), unicestwienie przeciwników (100pkt za *blind*, 200pkt za *random* i 300pkt za *charger*), ukończenie poziomu i wejście do portalu (1000pkt) + do punktów gracza dodaje się 10pkt * ilość pozostałego czasu gracza w sekundach. Jeżeli ilość punktów gracza przekroczy highscore, to do highscore zostaje przypisywana na bieżąco ilość punktów gracza.

2. Liczba pod tytułem "TIME" to czas, w jakim należy ukończyć mapę. Jeżeli ta wartość spadnie do zera, to gracz zacznie tracić 10% punktów na sekundę, a jak ilość punktów spadnie poniżej 10, to gracz przegrywa grę.

3. Każdy przeciwnik, jeżeli znajdzie się w polu rażenia eksplozji bomby, zostanie unicestwiony.

4. Jeżeli ulepszenie zostanie dotknięte przez przeciwnika lub znajdzie się ono w zasięgu eksplozji bomby, to zniknie.

5. Zebranie ulepszenia polega na wejściu gracza na kafelek zawierający ulepszenie,

6. Po zebraniu ulepszenia kopania bomb gracz może wejść na kafelek zawierający bombę, jeśli może ona być kopnięta, tj kafelek dalej znajduje się pusta przestrzeń - ścieżka. Kiedy bomba zostaje kopnięta, to porusza się zgodnie z kierunkiem kopnięcia aż do swojej eksplozji lub aż do napotkania przeszkody albo ognia (co sprawia, że bomba natychmiast wybucha).

7. Kiedy bomba znajdzie się w zasięgu eksplozji innej bomby, to również eksploduje.

8. Gracz przegrywa, gdy zostanie śmiertelnie zraniony przez przeciwnika, przez eksplozję bomby lub kiedy skończy mu się czas i punkty. Zmienia wtedy swój kolor na ciemnoczerwony i po chwili całe okno zajmuje okno pauzy z treścią:

   ```
   ''GAME OVER! PRESS SPACE TO RESTART
   SCORE XXXX'',
   ```

   gdzie XXXX to aktualna ilość punktów gracza. Po naciśnięciu klawisza spacji gra rozpocznie się na nowo, ulepszenia gracza zostaną zresetowane do wartości domyślnych, a jego ilość punktów wyzerowana.

9. Wygranie plaszy nastąpi, kiedy gracz pokona wszystkich przeciwników i wejdzie w portal na środku mapy, który się wtedy pojawi. Zacznie się wtedy przelewanie pozostałego czasu gracza w sekundach do aktualnych punktów gracza. Kiedy przelew się zakończy, to po chwili pojawi się okno pauzy z treścią :

   ```
   ''YOU WON! PRESS SPACE TO GO ON
   SCORE XXXX'',
   ```

   gdzie XXXX to aktualna ilość punktów gracza. Po naciśnięciu klawisza spacji gra rozpocznie się na nowo, a gracz zachowa swoje ulepszenia i punkty.

10. Highscore jest przechowywany w pliku "high score.txt" w formacie:

    ```
    ''HIGHSCORE: XXXX'',
    ```

    gdzie XXXX to wartość highscore.

11. Jeżeli gra zostanie włączona w trybie ”debugMode”, to gracz ma dodatkowo dostęp do następujących funkcji:
    (a) klawisz ”S” włączy lub wyłączy graczowi ulepszenie tarczy,
    (b) klawisz ”K” włączy lub wyłączy graczowi ulepszenie kopania bomb,
    (c) klawisz ”LSHIFT” zwiększy zasięg bomb gracza,
    (d) klawisz ”RSHIFT” zmniejszy zasięg bomb gracza,
    (e) klawisz ”LALT” zwiększy ilość dostępnych dla gracza bomb,
    (f) klawisz ”RALT” zmniejszy ilość dostępnych dla gracza bomb,
    (g) klawisz ”0” otworzy portal niezależnie od ilości aktywnych przeciwników.

4 Specyfikacja wewnętrzna

Pomocnicza specyfikacja wewnętrzna wszystkich plików, funkcji, enumeratorów, makr i struktur znajduje się w załączniku do sprawozdania. Ważniejsze zmienne lokalne zostaną opisane w następujących tabelach

plik **main.c**

enemyArray	zmienna typu **Enemy**[**MAX_ENEMIES_PER_MAP**]. Przechowuje dane wszystkich przeciwników. **MAX_ENEMIES_PER_MAP** to makro zdefiniowane w plik **enumsStructsMacros.h**. Określa maksymalną ilość przeciwników.
player	zmienna typu **Player**. Struktura przechowująca dane gracza. Dane te są inicjalizowane w funkcji **initPlayer()** w pliku **player.c**.
bitmap	zmienna typu **tile****. Wskazuje na dwuwymiarową tablicę, która reprezentuje planszę gry. Jej wielkość jest określana przez parametr funkcji **main** lub ustawiona jest domyślnie. Pamięć zostaje zaalokowana w funkcji **allocateMap()** w pliku **mapMemAndGen.c**.
pauseType	zmienna typu **PauseScreenType**. Przechowuje informację, w jakim aktualnie gra jest w stanie pauzy. Jej wartość jest zmieniana w funkcjach **pauseGame()** oraz **unPauseGame()**, obydwie są w pliku **main.c**.
listArray	zmienna typu **int** [**6**]. Tablica zawiera w sobie indeksy stworzonych w programie **glLists** (list instrukcji rysujących). Jest zapełniana w funkcji **setGlLists()**, w pliku **draw.c** oraz opróżniana w funkcji **unSetGlLists()**, również w pliku **draw.c**
portalOpened	zmienna typu **int**. Jeśli portal nie został jeszcze otworzony, to wynosi 0. W przeciwnym przypadku wynosi 1.
dTime	zmienna typu **double**. Przechowuje czas sprzed pauzy, aby można było do niego wrócić po wyjściu z pauzy. Zmiana jego wartości następuje w **pauseGame()**, a wykorzystuje się go do ustawienia czasu w funkcji **unPauseGame()**, obydwie są w pliku **main.c**.

plik **draw.c**

lastTime	zmienna statyczna typu **double**, znajdująca się w funkcji **checkPlayer()**. Przechowuje ostatni czas dokończenia okresowej animacji portalu.

plik **Enemies.c**

iteration	zmienna statyczna typu **int**, znajdująca się w funkcji **moveBlind()**. Kontroluje zmianę kierunku przeciwnika typu *blind*, np. dla **iteration = 0**, funkcja spróbuje poruszyć przeciwnika w lewo, a dla **iteration = 2**, zawróci go.
time_t	zmienna statyczna typu **time_t**, znajdująca się w funkcji **moveBlind()**. Służy do ustawienia ziarna generatora liczb psełdoprzypadkowych. Należy ustawić go tylko raz.
seedOnce	zmienna statyczna typu **int**, znajdująca się w funkcji **moveBlind()**. Upewnia się, że ziarno generatora liczb psełdoprzypadkowych (**srand(time(tt))**) zostanie ustawione tylko raz.
lastMoveTimeX, $X \in 1, 2, 3$	zmienna statyczna typu **double**, znajdująca się w funkcji **manageEnemiesAndScore()**. Przechowuje czas kiedy ostatni raz został ruszony dany typ przeciwnika. Dla X = 1, dla przeciwnika typu *blind*, X= 2 *random*, a dla X=3 *charger*.

plik **mapMemAndGen.c**

time_t	zmienna statyczna typu **time_t**, znajdująca się w funkcji **generateMap()**. Służy do ustawienia ziarna generatora liczb psełdoprzypadkowych. Należy ustawić go tylko raz.
seedOnce	zmienna statyczna typu **int**, znajdująca się w funkcji **generateMap()**. Upewnia się, że ziarno generatora liczb psełdoprzypadkowych (**srand(time(tt))**) zostanie ustawione tylko raz.

plik **player.c**

lastTime	zmienna statyczna typu **double**, znajdująca się w funkcji **checkPlayer()**. Można dzięki niej kontrolować zmiany spowodowane upływem czasu, bo przechowuje czas, kiedy ostatnia akcja została wykonana (np. kiedy ostatni raz odjęto punkty, żeby sprawdzić, czy nie powinno się znowu tego zrobić).
wait	zmienna statyczna typu **double**, znajdująca się w funkcjach **waitForGame_over()** oraz **waitForNextStageAndSumScore()**. Przechowuje czas rozpoczęcia wydarzenia (przegrania lub wygrania gry), dzięki czemu można sprawdzić, czy minęło dość dużo czasu, by zakończyć grę.

plik **TestArg.c**

badUse	zmienna typu **char***. Przechowuje wiadomość jaką należy wypisać do konsoli w przypadku złego użycia parametrów funkcji **main()**.
help	zmienna typu **char***. Przechowuje wiadomość jaką należy wypisać do konsoli w przypadku użycia przełącznika "-h" lub "-help" jako parametr funkcji **main()**.
zleWartosci	zmienna typu **zleWartosci**[**7**]. **zleWartosci**[i] przechowuje wartości wskazujące na rodzaj złej wartości i-tego parametru.1 dla wartości nierozpoznanej jako liczba, 2 dla wartości spoza zakresu.
zleWartosci	zmienna typu **int**[**7**]. **zleWartosci**[i] przechowuje wartości wskazujące na rodzaj złej wartości i-tego parametru funkcji **main()**. 1 dla wartości nierozpoznanej jako poprawna liczba, 2 dla wartości spoza zakresu.
ileRazyPrzelacznik	zmienna typu **int**[**7**]. **ileRazyPrzelacznik**[i] przechowuje ilość wystąpień i-tego przełącznika w parametrach funkcji **main()**.

5 Testowanie

Program został przetestowany dla róznych ilości przeciwników, pudełek, ulepszeń oraz dla różnych rozmiarów mapy. Nie znaleziono przy tym żadnych błędów, co wskazuje na to, że wszystkie zmienne są dobrze dopasowywane do siebie i kontrolowane wewnątrz programu.

6 Wnioski

Projekt ten nie zawiera bardzo skomplikowanych funkcji i algorytmów, ale ilość elementów do obsługi szybko zaplącze kod nieuważnego programisty. Aby iść do przodu, należało z dyscypliną dbać o porządek w kodzie i plikach i dzielić kod na sensowne i łatwe do zrozumienia funkcje, które wykonują tylko to, co sugeruje ich nazwa. Tylko wtedy praca, zamiast frustracji z powodu narastającego chaosu i błędów, przynosiła przyjemność.