Educational Technology
Образователни технологии

ОТ МАТЕМАТИЧЕСКИ КЪМ КОМПЮТЪРНИ ИГРИ

Павел Азълов
Пенсилвански държавен университет

Резюме. Едно от приложенията на компютърните игри е в областта на образованието. Този вид игри се използва както при усвояване на нови знания, така и при изграждане на умения за вземане на логически обосновани решения в непозната обстановка. Самото разработване на несложни логически игри е също с огромен образователен ефект. Това е и акцентът на тази статия.

Тръгвайки от конкретна математическа задача, описана като игра, в статията е представен модел за изграждане на множество компютърни игри върху грид. Изградено е ядро от необходимите програмни класове, с които се моделират средата за игра (грид), участниците в играта и самата игра, която ги обединява в едно цяло. На базата на две конкретни игри, конструирани от програмните класове, се показва начинът, по който могат да се създадат нови игри върху грид.

Всяка игра има специфична инициализация на грида, правила за „придвижването на фигурите“, критерий за завършване на играта и обявяване на победителя. Всеки един от тези елементи на играта е илюстриран чрез функциите, с които се реализират алгоритмите на две конкретни игри.

Формулирано е и множество от игри, които могат програмно да се разработят на базата на представения модел. Някои от тези игри имат изследователски характер и могат да бъдат в основата на интересни и нетривиални проекти.

Keywords: computer games, matematical games, games on grid

Вместо увод

Компютърните игри са част от ежедневието ни. Използват се предимно за развлечения, но много от тях имат и образователен характер. Масовото използване на компютърните игри от хора на всички възрасти мотивира софтуерните компании в разработването на софтуер за игри от всякакъв вид. Напоследък се провеждат конференции, отнасящи се до теорията и практиката на компютърните игри. Редица издателства предлагат книги по компютърни игри, а много университети въведоха курсове по компютърни игри (Leuteneger & Edgington, 2007).

И у нас, но още преди 40 години, в рамките на поредицата „АЛЕФ“ издателство „Народна просвета“ издаде книга със заглавие „Математически игри“ (Чуканов, 1975). В книгата са представени анализът и алгоритмите на разнообразни

игри, които са в основата на много от днешните компютърни игри. Автор на книгата е Владимир Чуканов, който за съжаление, напусна завинаги математическата ни колегия твърде млад. Едва ли Владимир се е надявал, че тази книга може бъде актуална след толкова много години. Книгата „Математически игри“ е вече почти забравена, а за по-младите читатели на списанието е непозната, но тя винаги е била всред книгите от личната ми библиотека. Всъщност идеята за настоящата статия е породена от книгата на Владимир Чуканов.

1. Математически и компютърни игри

Математически са тези игри, които се поддават на математически анализ. Може и по-дирекно да се каже, че математическата игра е математическа задача. За някои математически игри се казва, че са *логически*. В такива случаи те се използват и за развлечение. В интернет се среща и терминът „*игри за мислещи*“, като с това, явно, се подсказва, че основната цел в тях е да се играе с добре обмислени ходове, а не в „бързото кликване“ върху екрана на компютъра, което е типично за видеоигрите.

Математическите игри са *дискретни* и обикновено *крайни*. Това означава, че играта се „разбива“ на последователност от отделни ходове, които участващите извършват един след друг. Крайните игри винаги завършват след краен брой ходове на участниците. Примери за дискретни и крайни игри, които често играем, са „Кръстчета и нули“, НИМ (Чуканов, 1975), „Бикове и крави“ (Азълов & Златарова, 2011), (Grozdev & Doichev, 2005) и много други.

Всяка игра се определя с:

− обектите, с които се играе (камъчета, монети, моливи, шахматни фигури и др.);

− средата, в която се извършват ходовете на участниците (шахматна дъска, таблица, състояща се от редове и колонки върху лист хартия, и др.);

− системата от правила за извършване на ходове от участниците в играта. В системата има правило, с което се определя началната позиция (конфигурация) на обектите, с които се играе, правило, с което се определя кога настъпва краят на играта и кой от участващите в нея е победител, ако има такъв;

− броя на участниците в играта (един, двама или повече от двама).

Основна цел в математическите игри е формулирането на *стратегия за игра*. Системата от правила, които използва участникът в една игра, определят неговата стратегия. Естествено, всеки от участниците се стреми да приложи стратегия, която му осигурява победа. Такава стратегия, ако тя изобщо съществува, се нарича *печеливша*. Могат да се поставят редица въпроси, свързани със стратегията за победа, като например:

− Кои са условията, при които може да се формулира печеливша стратегия?

− Ако няма печеливша стратегия, възможно ли е да се формулира стратегия, при която да се постига равен резултат?

− Единствена ли е печелившата стратегия?

Това, както се вижда, са типични математически въпроси.

Най-общо може да се приеме, че текущата *позиция* в една партия се характеризира с броя и/или текущата позиция на обектите, участващи в играта в дадения момент. В някои игри е възможно да се определи дали тя е *благоприятна,* или е *неблагоприятна*.

Позицията е благоприятна за играча, който е на ход, ако той има печеливша стратегия, т.е. ако той може да спечели партията независимо от стратегията и конкретните ходове на другия играч. По същество това означава, че една позиция е благоприятна за единия играч, ако всеки път, когато е на ход, **съществува поне един такъв ход**, който я обръща в неблагоприятна за другия играч. Може да се каже също, че ако **всеки възможен ход** превръща дадена текуща позиция в благоприятна за другия играч, то тази позиция е неблагоприятна за играча, който е на ход.

Формулирането на стратегия обикновено започва с анализирането на няколко „прости“ партии на играта. Това е начин да се търси определена математическа зависимост. Ако такава зависимост е намерена, тогава се формулира хипотеза за печеливша стратегия. Естествено, хипотезата трябва трябва да се докаже. Това е идеалният случай, но той невинаги е възможен.

Компютърните игри са твърде разнообразни и класифицирането им е сложно. Затова е добре да се уточни, че в тази статия понятието „*компютърна игра“* трябва да се разбира в по-тесен смисъл на този термин. Тук компютърната игра е компютърна програма, с която е реализирана идея от някаква игра. Играта може да бъде математическа и за нея да има или да няма доказана печеливша стратегия. Ако такава съществува, тогава компютърът ще е единият от участниците в играта, който ще прилага печелившата стратегия. Ако все още няма формулирана печеливша стратегия, проследяването на развоя на изиграните игри би могло да подпомогне формулирането на такава стратегия, ако тя изобщо съществува. Играта може и да не се поддава на пълен математически анализ, но да има алгоритмичен характер. Това означава, че целта в нея е да се построи алгоритъм, който да дава някакво (не непременно оптимално) решение на задачата, формулирана в играта. Ако ходовете на играча, които се моделират с алгоритъма, се приемат за „разумни“, то целта е постигната.

Подходът, който е използван за изграждането на една компютърна игра, е илюстриран за игри върху правоъгълна мрежа. За да се улесни изпълнението им, ходовете на всеки от играчите се извършват програмно. Това означава, че един

от играчите ще използва доказана печеливша стратегия за игра (ако такава съществува), а ходовете на останалите играчи ще се извършват по случаен начин от компютъра.

2. Пример на математическа игра

Игра с бели и черни бобчета

Формулировка на играта. Дадени са две купчини с бобчета – едната с бели, а другата с черни. Участват двама играчи, които извършват последователно по един ход. При всеки ход играчът може да вземе само едно бобче (бяло или черно) или две бобчета, но с различен цвят. Играта завършва, когато и последното бобче е взето. Победител е играчът, след чийто ход не остава нито едно бобче.

Анализ на играта и хипотеза за печеливша стратегия. Приемаме, че партията започва с M бели и N черни бобчета, като това означаваме с (M, N). Очевидно, играта е симетрична по отношение на белите и черните бобчета, т.е. (M, N) и (N, M) са една и съща партия. Цветът е въведен само за да улесни обясненията в анализа на играта. За удобство ще означим с A и B съответно първия и втория играч. Един играч е първи, ако той прави първия ход.

Да анализираме няколко партии с малки стойности на M и N (табл. 1). Естествено е да се приеме по дефиниция, че позицията (0, 0) е неблагоприятна, защото не е възможно да се направи следващ ход. За всяка от партиите в таблицата може директно да се провери дали началната позиция е, или не е благоприятна за играча A.

Таблица 1. Благоприятни и неблагоприятни позиции за различни стойности на M и N

1	$M = 0, N = 0$	(0, 0) – неблагоприятна позиция за A	M - четно	N - четно
2	$M = 1, N = 0$	(1, 0) – благоприятна позиция за A	M - нечетно	N - четно
3	$M = 1, N = 1$	(1, 1) – благоприятна позиция за A	M - нечетно	N - нечетно
4	$M = 2, N = 0$	(2, 0) – неблагоприятна позиция за A	M - четно	N - четно
5	$M = 2, N = 1$	(2, 1) – благоприятна позиция за A	M - четно	N - нечетно
6	$M = 3, N = 0$	(3, 0) – благоприятна позиция за A	M - нечетно	N - четно
7	$M = 2, N = 2$	(2, 2) – неблагоприятна позиция за A	M - четно	N - четно
8	$M = 3, N = 1$	(3, 1) – благоприятна позиция за A	M - нечетно	N - нечетно
9	$M = 3, N = 2$	(3, 2) – благоприятна позиция за A	M - нечетно	N - четно

Да разгледаме последните две колонки на таблицата (табл. 1). От тях може да се забележи, че играчът A е в неблагоприятна позиция само когато M и N са едновременно четни числа. Това подсказва идея за формулиране на хипотеза за печеливша позиция: играчът A е в печеливша позиция, когато и двете числа M и N не са едновременно четни. Тази хипотеза трябва да се докаже. Ако тя е вярна, тогава печелившата стратегия на играча A се свежда до вземането на бобчета по такъв начин, че винаги във всяка купчинка да остават четен брой бобчета. Неговият печеливш ход ще бъде един от следните три възможни хода ($M > 0$, $N > 0$):

- ако M е нечетно и N е нечетно → $(M - 1, N - 1)$;
- ако M е нечетно и N е четно → $(M - 1, N)$;
- ако M е четно и N е нечетно → $(M, N - 1)$

Позицията, при която M е четно и N е четно, е неблагоприятна за A. Всеки един от ходовете $(M - 1, N - 1)$, $(M - 1, N)$ и $(M, N - 1)$ е благоприятна позиция за играча B и той може да победи, ако знае как да се възползва от това.

Доказателство на хипотезата. Изказаната по-горе хипотеза е вярна и доказателството може да извърши чрез индукция по сбора $M + N$, така както това е направено в (Чуканов, 1975) за играта „Еднопосочен цар“, формулирана по следния начин:

„*В горния десен ъгъл на шахматна дъска с размери $m \times n$ е поставен цар. Играят двама играчи, които последователно местят царя. Общо взето, царят се движи като обикновен шахматен цар, но с известно ограничение в движението си – може да се премества само надолу, наляво и по диагонала надолу-наляво. Печели играчът, който пръв постави царя в долния ляв ъгъл на дъската*.“

Тази игра е еквивалентна на играта „Бели и черни бобчета“. Придвижването на царя наляво съответства на вземане на бобче от едната купчина, придвижването му надолу – на вземане на бобче от другата купчина, а придвижването му по диагонал – на вземане на две бобчета, едно бяло и едно черно.

Алгоритъм на играта. Алгоритъмът, приведен по-долу, е ориентиран към играча A, който познава играта и може да избира правилно печелившите си ходове.

1. [*Начало на играта*] Начална позиция на играта. Посочват (въвеждат) се стойностите на M и N ($M \geq 0$, $N \geq 0$ и $M + N > 0$).

2. [*Основен цикъл на играта*] Докато $M + N > 0$, се повтарят следните действия:

2.1 [*Ход на A*] Пресмятане на нов ход за A.

Ако поне едно от числата M и N е нечетно (печеливша позиция), избира се съответният правилен ход от разгледаните по-горе три възможни хода. В противен случай се прави произволен ход.

2.2 [*Проверка за край на играта*] Ако $M = 0$ и $N = 0$, победител е играчът A. Изпълнява се т. 3.

2.3 [*Ход на B*] Пресмятане на нов ход за играча *B*.

2.4 [*Проверка за край на играта*] Ако *M* = 0 и *N* = 0, победител е играчът *B*. Изпълнява се т. 3.

3. [*Край на играта*] Обявяване на победителя.

3. Преход от математическа към компютърна игра

По-долу е дадена програмната структура на играта „Бели и черни бобчета“, представена с прототипите на основните функции, от които тя е изградена (фиг. 1), както и текста на две функции (*main*, *gameEngine*), които реализират развитието на играта. В играта участват играчите *A* и *B*, от които *A* използва представената по-горе стратегия, а ходовете на *B* се избират по случаен начин. В реална игра ходовете на *B* се въвеждат от съответния играч.

```
int gameEngine(int, int);          // „Двигател“ на играта
void strategy(int&, int&);         // Избор на благоприятна позиция
void randMove(int&, int&);         // Избор на случаен ход
bool isEndGame(int, int);          // Проверка за край на играта
void winner(int);                  // Обявяване на победителя
void printMove(int, int, int);     // Печат на пореден ход от играта

int main()
{
  srand(unsigned int(time(0)));
  int m, n;                        // Брой на бобчетата (m, n)
  cout << "Enter values for M and N: ";
  cin >> m >> n;
  winner(gameEngine(m, n));
  return 0;
}

int gameEngine(int m, int n)
{
   int move = 1;
   cout << "Start ";
   printMove(0, m, n);
   while (true)                    // Основен цикъл на играта
   {
      cout << "\nMove " << move << endl;
```

```
        move++;
        strategy(m, n);                    // Ход на първия играч (A)
        printMove(1, m, n);
        if (isEndGame(m,n)) return 1;      // Победител е A

        randMove(m, n);                    // Ход на втория играч (B)
        printMove(2, m, n);
        if (isEndGame(m,n)) return 2;      // Победител е B
    }
}
```

Фигура 1. Структура на програмата,
реализираща играта „Бели и черни бобчета“

Следват три партии (табл. 2), в които отделните ходове на играчите се извършват от компютъра. Началната позиция в първата партия е благоприятна за играча A, който позназва анализа на играта и е естествено той да победи. В останалите две партии началната позиция не е благоприятна за A и резултатът зависи от ходовете на играча B.

Таблица 2. Три изпълнения на програмата, реализираща играта
„Бели и черни бобчета“

Enter values for M and N: 3 6 Start: [M=3, N=6] Move 1 Player 1: [2, 6] Player 2: [1, 6] Move 2 Player 1: [0, 6] Player 2: [0, 5] Move 3 Player 1: [0, 4] Player 2: [0, 3] Move 4 Player 1: [0, 2] Player 2: [0, 1] Move 5 Player 1: [0, 0] The winner is A	Enter values for M and N: 2 6 Start: [M=2, N=6] Move 1 Player 1: [1, 5] Player 2: [0, 4] Move 2 Player 1: [0, 3] Player 2: [0, 2] Move 3 Player 1: [0, 1] Player 2: [0, 0] The winner is B	Enter values for M and N: 4 6 Start: [M=4, N=6] Move 1 Player 1: [3, 5] Player 2: [2, 4] Move 2 Player 1: [1, 3] Player 2: [0, 3] Move 3 Player 1: [0, 2] Player 2: [0, 1] Move 4 Player 1: [0, 0] The winner is A

Основен цикъл на компютърна игра

Да разгледаме по-внимателно „двигателя на играта“ (функцията *gameEngine*), която осъществява последователността на ходовете на отделните играчи (фиг. 2). Ако се освободим от допълнителните оператори, които се използват за отпечатване на изиграните ходове, се получава следният код:

```
int gameEngine(int m, int n)
{
   while (true)                          // Основен цикъл на играта
   {
      strategy(m, n);                    // Ход на първия играч (A)
      if (isEndGame(m,n)) return 1;      // Победител е A
      randMove(m, n);                    // Ход на втория играч (B)
      if (isEndGame(m,n)) return 2;      // Победител е B
   }
}
```

Фигура 2. Управляваща функция на играта

Цикълът в тялото на функцията се нарича *основен цикъл на играта*. В общия случай в една игра могат да участват p играчи ($p \geq 1$) и тогава основният цикъл може да се запише така (фиг. 3):

```
while (true)                             // Основен цикъл на играта
{
   for (int id=1; id<=p; id++)           // Цикъл по всеки играч
   {
     move(game, id);                     // Ход на играч с номер id
     if (isEndGame(game)) return id;     // Проверка за край на играта
   }
}
```

Фигура 3. Основен цикъл на компютърна игра

Процедурата *move* извършва ход съгласно стратегията на играча с номер *id* в играта *game*. Първият аргумент (*game*) на процедурата е обект, в който се съхранява цялата информация от текущото състояние на играта. Това са данни за всички играчи и за средата (игралната площадка), върху която се играе.

Структура на компютърна игра

Разгледаната игра е твърде елементарна, но от програмната ѝ реализация могат да се видят основните *функционални компоненти*, характерни за много от компютърните игри:

− стартиране на играта – функция, с която се инициализират параметрите на играта (игрална площадка, брой играчи и др.), представят се правилата на играта и др.;

− „двигателя на играта“ – функция, извършваща последователността от ходовете на отделните играчи;

− стратегия за игра – функция, с която се избира всеки следващ ход на даден играч;

− критерий за прекратяване на играта – функция, с която се проверява за настъпването на края на играта;

− визуализиране на последователността от ходовете, извършени по време на една партия;

− оповестяване края на играта и на резултата от нея.

Към списъка от функции, които се извършват в компютърните игри, ще добавим и основните *структурни компоненти* на играта. Това са програмни структури (структури от данни, модули и класове), с които се представят:

− характеристиките на средата, върху която се играе, и на операциите, които могат да се извършват върху нея;

− характеристиките на участник в играта и на операциите, които той може да извършва;

− параметрите и правилата на игра.

Грид – „Сцена за игра“

Във формулировката си на „Еднопосочен цар“ играта „Бели и черни бобчета“ придобива по-забавен вид и се доближава до много други игри, използващи като игрална площадка множество от клетки, структурирани в редове и колонки. Като имена на такава структура в игрите се използват термините шахматна дъска, таблица, квадратна решетка и грид. И въпреки че в играта „Еднопосочен цар“ става дума за шахматна фигура, то цветът на клетките в шахматна дъска не е от значение. Поради това по-нататък ще бъде използван краткият и по-общ термин *грид* като сцена, на която всеки от играчите ще премества своята фигура (шахматен цар, топ, царица, кон или някаква специална фигура-робот със съответни правила на придвижване) от една в друга клетка (поле) на грида. За разлика от шахматната дъска размерите на грида могат да бъдат произволни цели положителни числа, а клетките да бъдат маркирани като достъпни или недостъпни за фигурите.

Общ сценарий за игри върху грид

Представената игра и нейната програмна реализация са един опростен, но добър пример, който може да бъде в основата на програмен проект за разработване на компютърни игри върху грид.

Игра върху грид. Компонентите на една игра върху грид са: грид, играчи и правила за игра, за които се приема, че:

− гридът е таблица с $M \times N$ клетки, структурирани по редове и колонки. Той може да е празен. В този случай всяко поле е достъпно за фигурите, участващи в играта. В някои игри е възможно върху грида да има недостъпни полета. Идея за това ни дават игрите, в които се използват лабиринти;

− броят на играчите е един, двама или повече;

− всяка от фигурите е определена с правилата, с които се извършва придвижване (преход) от една клетка в друга клетка на грида. В системата от правила се включва инициализирането на играта, т.е. определят се размерите и съдържанието на грида, началните позици на всяка от фигурите, както и позицията, при която играта завършва.

Тук трябва да се направят две важни бележки.

− понятията фигура и играч, който играе с фигурата, се припокриват. Не е важно каква е фигурата. Тя се движи (премества) по определени правила, които играчът знае и прилага, за да направи следващия си ход;

− терминът „придвижване на фигура“ трябва да се приема абстрактно и всъщност означава ход на играча, който играе с тази фигура. В играта „Еднопосочен цар“ фигурата се премества от едно в друго поле на шахматната дъска, но в играта „XO“ („Кръстчета и нули“) няма движение на фигура, а само маркиране на клетка със съответния знак на играча („O“ или „X“ при игра с двама играчи).

Модел на игра върху грид

Разглеждаме произволна игра G върху грид. Разположението на фигурите върху грида в конкретен момент от играта се нарича *състояние на играта*. Означаваме го с s, а множеството на всички състояния, наречено *пространство на състоянията*, означаваме с S.

Всяка игра започва с някакво *начално* състояние $\underline{s}$, $\underline{s} \in S$. Играта завършва при достигане на състояние, наречено *крайно* $\underline{s}$, $\underline{s} \in S$. За някои игри крайното състояние може да не е единствено.

Ход в играта е действие, което привежда играта от едно състояние s в ново състояние s'. С други думи, всеки ход h, приложен към текущото състояние s на играта, я привежда в ново състояние $\underline{s}$, $\underline{s} \in S$. Това може да се запише като *функция на преход* по следния начин:

$s' = \Im(s, h)$

Произволна *партия* от играта G е крайна редица от ходове, започваща с начално и завършваща с едно от възможните крайни състояния:

G: $\underline{\boldsymbol{s}} \overset{h}{\rightarrow} s_1 \overset{h}{\rightarrow} s_2 \ldots \overset{h}{\rightarrow} \overline{\boldsymbol{s}}$

В зависимост от броя на играчите отделните ходове h се извършват от един или от няколко играчи в последователен ред.

4. Изграждане на програмни модули за игри върху грид

На базата на примерната игра „Еднопосочен цар“ и на общия модел на игри върху грид могат да се изградят основните програмни структури, които да бъдат ядро за реализиране на множество от *конкретни игри върху грид*. За тази цел ще бъдат проектирани и реализирани четири модула, всеки от които, определен със съответните данни и операции, ще представлява компонент от игра върху грид. В обектноориентираните езици, какъвто е и езикът C++, това се постига чрез използване на класове.

По-долу следва краткото представяне на тези четири класа. За по-лесно разбиране на предназначението на функциите имената им в тези класове са унифицирани. С думата *get* започват имената на функциите *селектори*, т.е. функциите, които връщат текущи стойности на величини от *private* секцията на класовете, а с думата *set* започват имената на функциите *мутатори*, т.е. функциите, с които се променят текущите стойности на величините от *private* секцията на класовете. Имената на функциите на класа подсказват предназначението им, което се допълва от краткия коментар, следващ декларацията им.

Класът *Grid*

Грид в игрите, които се разглеждат, е двумерен масив от клетки. Всяка клетка е определена от координатите си. Приема се, че координатите на отделната клетка са цели неотрицателни числа, които са координатите на долния ляв връх на клетката (квадратчето). Един пример е даден на фиг. 4. Декларацията на клетка е дадена със структурата Coord. Освен координатите на клетката в декларацията присъстват и три често използвани операции, определени с функциите: *конструктор*, *сравняване за равенство на две клетки* и *отпечатване*.

2	(0,2)	(1,2)	(2,2)	(3,2)
1	(0,1)	(1,1)	(2,1)	(3,1)
0	(0,0)	(1,0)	(2,0)	(3,0)
	0	1	2	3

Фигура 4. Грид с размери 3x4

На фиг. 5 е дадена декларацията на структурата *Coord* и на класа *Grid*. Величините *MX* и *NX* са глобални константи, дефинирани в отделен модул.

```
struct Coord
{
  int x, y;
```

```
  Coord (int=0, int=0);
  bool operator==(Coord) const;       // Проверка за равенство
                                         на две клетки
  void print();
};

class Grid
{
  public:
    Grid();                           // Конструктор по премълчаване
    Grid(int, int, int=0);            // Инициализиране на всички
                                         данни
    ~Grid();                          // Деструктор

    void setDimX(int);                // Определя броя на редовете
    void setDimY(int                  // Определя броя на колонките
    int getDimX() const;              // Връща броя на редовете
    int getDimY() const;              // Връща броя на колонките

    void setTable(int, int, int=0);   // Присвоява стойност на
                                         клетките на грид
    void setInaccCells(int);          // Недостъпни клетки
    void setTable(int[]);             // Определя грид по даден масив
    void getTable(int[]);             // Връща масив, съдържащ грид

    void setCell(Coord, int);         // Присвоява стойност на клетка
                                         от грид
    int getCell(Coord) const;         // Връща стойност на клетка
                                         от грид
    bool inSide(Coord) const;         // Проверка дали клетката е
                                         вътрешна за грид

    void print() const;               // Отпечатва размерите на грид
    void printTable() const;          // Отпечатва таблицата на грид

  private:
    int dimX;                         // Брой редове на грида
    int dimY;                         // Брой колонки на грида
```

```
    int table[MX*NX];              // Грид с размери dimX x dimY
    int index(int, int) const;     // Трансф. на двумерен в
                                      едномерен масив
};
```

Фигура 5. Декларация на структурата Coord и на класа *Grid*

Всеки обект от класа *Grid* е таблица с текущи размери *dimX*, *dimY* и е представен с масива *table*. За двумерните масиви в C++ се изисква предварително определяне на размера на реда, което създава известни неудобства. За да не се утежнява програмният текст, вместо двумерен динамичен масив от тип *vector* е използван едномерен масив (*table*), който чрез функцията *index* се използва като двумерен.

Класът *Player*

Обектите на класа *Player* са участниците в играта. Като променливи те са представени с две величини – име и съответни текущи координати (позиция) върху грида (фиг. 6).

```
class Player
{
  public:
    Player();                  // Конструктор по премълчаване
    Player(string, Coord);     // Инициализиране на всички данни
    Player(const Player&);     // Copy-конструктор
    ~Player();                 // Деструктор

    void setPlayer(Player);    // Присвоява данните на играч
                                  (име, позиция)
    void setCurrPos(Coord);    // Присвоява текуща позиция на играч
    Coord getCurrPos() const;  // Връща текущата позиция на играч
    void setName(string);      // Присвоява име на играч
    string getName() const;    // Връща име на играч

    void print() const;        // Отпечтва данни за играч

  private:
    string name;               // Име на играч
    Coord currPos;             // Позиция на играч
};
```

Фигура 6. Декларация на класа *Player*

Класът *Game*

Класът *Game* обединява грида и играчите в една игра. Всеки обект на класа съдържа един обект от клас *Grid* и няколко обекта от клас *Player* (фиг. 7). Това означава, че различните игри върху грид могат да имат различен брой играчи.

```
class Game
{
  public:
    Game();                                  // Конструктор по премълчаване
    Game(Grid, Player[], int, int);          // Инициализиране на всички данни
    Game(const Game &);                      // Copy-конструктор
    ~Game();                                 // Деструктор
    void play();
    const Game& operator=(const Game& pl); // Оператор
                                             за присвояване

    void setGrid(Grid gr);                   // Определяне на грид
    Grid getGrid() const;                    // „Връща“ текущия грид
    Player* getPlayers() const;              // „Връща“ данните на всички играчи
    Player getPlayer(int) const;             // „Връща“ данните на един играч
    void setPlayer (Player, int);            // Определя данните на един играч
    void setPlayers(Player*);                // Определяне на всички играчи
    void setNrPlayers(int);                  // Определяне броя на играчите
    int getNrPlayers() const;                // „Връща“ броя на играчите
    void setMaxNrPlayers(int);               // Определяне на max брой играчи
    int getMaxNrPlayers() const;             // „Връща“ max брой играчи
    void print();                            // Отпечатване на данните на играта

  private:
    Grid g;                                  // Грид на играта
    int nrPlayers;                           // Брой играчи
    int maxNrPlayers;                        // Максимален брой играчи
    Player* p;                               // Масив от играчи
};
```

Фигура 7. Декларация на класа *Game*

Класът *GameAlg*

Първите три класа изграждат обекти за игра върху грид, без да се специфицират нито правилата на игра, нито алгоритмите, които играчите използват, за да направят своя следващ ход. Правилата за игра и стратегиите за избор на ход

са представени чрез функциите в класа *GameAlg* (фиг. 8). Някои от тези функции са общи за игрите върху грид. Останалите функции представят специфичните правила на конкретната игра, както и стратегиите на участващите играчи. Това означава, че при реализирането на нова игра в тези функции трябва да се направят известни промени или да се напишат отново.

```
class GameAlg
{
  public:
    // Прототипи на функциите, специфични за играта „Ограничен цар“
    Game game1Init(int);              // Инициализиране на игра 1
    int game1Engine (Game);           // „Двигател“ на игра 1
    void game1nextMove (Game& game, int);   // Следващ
                                              ход на играч
    void game1P0nextMove (Game&);     // Следващ ход на първия играч
    void game1P1nextMove (Game&);     // Следващ ход на втория играч
    bool game1End (Coord) const;      // Проверка за край на игра 1
    void game1Winner (int) const;     // Обявяване на резултата
                                              от игра 1
    void print1Move(int p, Coord c);  // Печат на всеки ход 1

    // Прототипи на функциите, специфични за играта „Преследване“
    Game game2Init(int);              // Инициализиране на игра 2
    int game2Engine (Game);           // „Двигател“ на игра 2
    void game2nextMove (Game& game, int);   // Следващ ход
                                              на играч
    void game2P0nextMove (Game&);     // Следващ ход на първия играч
    void game2P1nextMove (Game&);     // Следващ ход на втория играч
    bool game2End (Coord, Coord) const; // Провека за край
                                              на играта
    void game2Winner (int) const;     // Обявяване на резултата
                                              от игра 2
    void print2Move(int p, Coord c);  // Печат на всеки ход 2
};
```

Фигура 8. Декларация на класа *Game*

Няколко от функциите в класа *GameAlg* са от особена важност и това е причината за представянето на текстове им.

Инициализиране на играта

Инициализирането на играта се извършва с отделна функция и включва определянето на размерите на грида, както и на броя, имената и началните координати на играчите. При играта „Еднопосочен цар“ началният ход и на двамата играчи е в горния десен ъгъл на грида. Тези начални данни определят началното състояние на обекта *game*, което функцията връща на главната функция.

```
Game GameAlg::game1Init(int inacc)
{
    srand(unsigned int(time(0)));
    int m = MM + rand() % (MX - MM); // Число в интервала [MM, MX]
    int n = NM + rand() % (NX - NM); // Число в интервала [NM, NX]
    Grid grid(m, n);                 // Инициализира се грид
                                       с размери m x n
    /*****************************************************************************************
    * Този блок трябва да се добави за игрите, в които гридът съдържа недостъпни клетки
    * int nrCell = m*n*INACCESSIBLE/100;  // Брой недостъпни клетки
    * if (nrCell > 0) grid.setInaccCells(nrCell);// Грид с недостъпни клетки
    *****************************************************************************************/
    int nrPlayers = 2;               // Брой на играчите
    Player player[MAXNUM_PLAYERS];   // Конструиране на
                                       обектите-играчи
    player[0].setName("P0"); player[0].
setCurrPos(Coord(m,n));
    player[1].setName("P1"); player[1].
setCurrPos(Coord(m,n));

    // Конструиране на обект от клас Game
    Game game(grid, player, nrPlayers, MAXNUM_PLAYERS);
    return game;
 }
```

Фигура 9. Функция за инициализиране на игра

„Двигател“ на играта

Вариант на тази функция е представен във фиг. 2, а по-долу е приведена реализацията ѝ за играта „Еднопосочен цар“. Идентификатори на играчите са техните номера, съответстващи на реда, в който те извършват пореден ход. Първият играч е с номер 0, вторият е с номер 1 и т.н.

```
  int GameAlg::game1Engine(Game game)
  {
     int nr = game.getNrPlayers();        // Брой на играчите
     int move = 1;
     while (true)                         // Основен цикъл на играта
     {
       cout << "\nMove " << move++ << endl;
       for (int id = 0; id < nr; id++)    // Цикъл за всеки отделен играч
       {
         game1nextMove(game, id);         // Ход на играч  с номер id
         print1Move(id, game.getPlayers()[id].
getCurrPos());   // Печат на ход

         // Проверка за край на играта
         if ( game1End (game.getPlayers()[id].
getCurrPos()) ) return id;
       }
     }
  }
```

Фигура 10. Функцията „Двигател“ на играта „Еднопосочен цар“

Избор на следващ ход

Предназначението на функцията *nextMove* е да управлява поседователността на ходовете на играчите (фиг. 11). В случая на играта „Еднопосочен цар“ играчите са двама и техните ходове се избират от функциите *game1P0nextMove* за първия играч и *game1P1nextMove* за втория играч. Първият играч е компютърът, който познава и използва печелившата стратегия.

```
  void GameAlg::game1nextMove (Game& game, int id)
  {
      // Номер на играча, играл последен
      int inx = (id + game.getNrPlayers() - 1) % game.
getNrPlayers();
      switch (id)
      {
         case 0: game1P0nextMove(game); break;
         case 1: game1P1nextMove(game); break;
         // Добавяне на трети играч, ако играта се играе от трима
         // case 2: game1P2nextMove(game); break;
      }
```

```
        // „Информиране“ на другия играч за изиграния ход
        game.getPlayers()[inx].setCurrPos(game.getPlayers()
[id].getCurrPos());
   }
```

Фигура 11. Избор на следващ ход

Стартиране на играта

Стартирането на играта започва от главната функция (фиг.12). Тя създава обект от класа *Game*, с който се стартира управляващата функция на играта *play*. Най-напред се създава обект от клас *GameAlg*, следва създаването на обект *game1*, инициализиран за играта „Еднопосочен цар“, както и отпечатването на кратка информация за играта и нейните участници. Играта започва с обръщение към функцията *game1Engine* и завършва с отпечатване на крайния резултат от играта.

```
   int main()
   {
     Game game;
     game.play();
     return 0;
   }
   void Game::play()
   {
      GameAlg gameAlg;
      Game game1 = gameAlg.game1Init(0);       // Конструиране на
обект от класа Game
      game1.print();                           // Печат на инфор-
мация за игра 1
      int win = gameAlg.game1Engine(game1);    // Старт на играта
      gameAlg.game1Winner(win);                // Обявяване на
победителя
    }
```

Фигура 12. Стартиране на играта

Таблица 3 илюстрира последователността от ходове на три партии.

Таблица 3. Три изпълнения на играта „Еднопосочен цар“

The game is starting	The game is starting	The game is starting
Number of players = 2 Grid’s dimensions = 5 x 5 Player name: P0 Initial position: [4, 3]	Number of players = 2 Grid’s dimensions = 6 x 8 Player name: P0 Initial position: [6, 8]	Number of players = 2 Grid’s dimensions = 5 x 5 Player name: P0 Initial position: [4, 3]

Player name: P1 Initial pos: [4, 3] Move 1 Player 0: [4, 2] Player 1: [4, 1] Move 2 Player 0: [4, 0] Player 1: [3, 0] Move 3 Player 0: [2, 0] Player 1: [1, 0] Move 4 Player 0: [0, 0] The winner is P0 The game is over	Player name: P1 Initial pos: [6, 8] Move 1 Player 0: [5, 7] Player 1: [4, 6] Move 2 Player 0: [3, 5] Player 1: [2, 5] Move 3 Player 0: [2, 4] Player 1: [1, 3] Move 4 Player 0: [0, 2] Player 1: [0, 1] Move 5 Player 0: [0, 0] The winner is P0 The game is over	Player name: P1 Initial pos: [4, 3] Move 1 Player 0: [4, 2] Player 1: [3, 2] Move 2 Player 0: [2, 2] Player 1: [1, 1] Move 3 Player 0: [0, 0] The winner is P0 The game is over

5. Примери на игри върху грид

Познати са разнообразни игри върху грид. По-долу следва кратък списък на такива игри, групирани в три категории.

Игри с шахматни фигури

G1.1: [*Еднопосочен цар*] Това е играта, която е описана в т. 3.

G1.2: [*Еднопосочна дама*] Играта е подобна на играта G1.1, но фигурата за игра е шахматна царица.

G1.3: [*Еднопосочен топ*] Играта е подобна на играта G1.1, но фигурата за игра е шахматен топ.

Бележка. Съществува математически анализ за игрите G1.2 и G1.3, с който могат се определят благоприятните позиции върху грида [3].

G1.4: [*Еднопосочен кон*] Играта е подобна на играта G1.1, но фигурата за игра е шахматен кон. Играта завършва, когато някой от играчите премести коня в едно от полетата с координати: (0, 0), (0, 1), (1, 0) и (1, 1).

G1.5: [*Еднопосочен цар*] Това е играта G1.1, която се играе с трима играчи.

G1.6: [*Еднопосочна дама*] Това е играта G1.2, която се играе с трима играчи.

G1.7: [*Еднопосочен топ*] Това е играта G1.3, която се играе с трима играчи.

G1.8: [*Еднопосочен кон*] Това е играта G1.4, която се играе с трима играчи.

Път върху грид: всички полета от грида са достъпни

G2.1: [*Най-кратък път*] Да се придвижи шахматен цар от една до друга клетка на грида с минимален брой ходове. В играта участва един играч.

G2.2: [*Преследване*] Фигурата *A* преследва фигура *B* върху грид. Достъпни за всяка от тях са 8-те съседни клетки на грида. Фигурата *B* не вижда *A* и не знае, че е преследвана, и се движи хаотично.

G2.3: [*Преследване*] Фигурата *A* преследва фигурата *B* върху грид. Достъпни за всяка от тях са 8-те съседни клетки на грида. Фигурата *B* „вижда“ фигурата *A* и иска да „избяга“ от нея.

G2.4: [*Преследване*] Играта е вариант на играта G2.3, но се играе с две фигури *A*1, *A*2, които независимо една от друга преследват фигурата *B*.

G2.5: [*Преследване*] Играта е вариант на играта G2.4, но фигурите *A*1, *A*2 играят в коалиция при преследването на фигурата *B*.

Забавни игри

G3.1: [*Играта „Кръстчета и нули“*] Играта е известна и с имената си „ХО“ и „Tic-Tac-Toe“. В класическия вариант тя се играе от двама играчи върху грид с размери 3 x 3, но може да се играе и от повече играчи върху грид с по-големи размери[1].

G3.2: [*Играта „Миночистач“* (“*Minesweeper*”)] Миночистач е игра, в която трябва да се търсят мини, разположени в клетките на грид с размери $M \times N$. Клетките на грида могат да бъдат празни или „минирани“. Съдържанието на грида се генерира по случаен начин (тайно). Чрез посочване на координатите на клетки от грида играчът се стреми да открие всички празни клетки, така че в края да останат само „минираните“ клетки. Ако е посочена „минирана“ клетка, той губи играта. Ако е посочена празна клетка, като отговор се получават координатите на всички съседни клетки, в които има мини. Съседни на дадена клетка са 8-те клетки, разположени по хоризонтална и вертикална линия, а също и по двата диагонала. Играта се играе от един играч.

G3.3: [*Играта „Миночистач“ за двама играчи*] Играта G3.2 може да се играе и от двама играчи, които последователно един след друг посочват клетки от грида. Губи този, който „уцели“ мина. Ако играта завърши, без да е „зривена“ мина, тя се печели от този, който е открил повече минирани клетки.

Това, което е общо за игри от този списък, е, че те са игри върху грид и могат да се реализират като компютърни игри чрез класове *Grid*, *Player*, *Game* и *GameAlg*.

Това, което ги различава, е:

− размерите на грида и началното му съдържание. Размерите могат да бъдат фиксирани или да са случайни числа от някакъв интервал. Гридът може да е „празен“ или върху него да има посочени „недостъпни“ клетки;

− броят на играчите и тяхната началната позиция. Броят на играчите може да е 1, 2 или повече. Началната позиция може да е фиксирана (G1.1, горният десен ъгъл) или избрана по произволен начин, което е характерно за повечето игри;

− множеството от правила за извършване на ход;

− условието, при което играта завършва, и определянето на победителя (евентуално равен резултат).

Не е случайно, че функциите, специфични за играта „Еднопосочен цар“, са дефинирани в отделен клас (*GameAlg*). Това означава, че за реализирането на нова игра функциите от този клас трябва да се напишат отново или само да се направят известни корекции на съответните съществуващи функции. Това е илюстрирано в следващата точка.

6. Нова компютърна игра

Ще разгледаме и реализираме една компютърна игра, именувана по-горе с G2.3. Това ще бъде пример за построяване на нова игра с помощта на четирите базови модула на играта „Еднопосочен цар“.

Помощна задача

Ще започнем с една помощна задача, формулирана като игра (G2.1). Да се намери път с минимална дължина между две произволни клетки от грида.

Път между две клетки *A* и *B* от произволен грид е редица от съседни клетки, която започва от *A* и завършва в *B*. Съседни на една клетка са всичките (не повече от 8) клетки, с които тя има обща страна или връх. Дължината на пътя се измерва с броя на клетките, започвайки от *A* и завършвайки в *B*.

Решение. Ако началната и крайната клетка съвпадат, тогава пътят е с дължина 0. Ако началната клетка (*start*) и крайната клетка (*end*) са върху един и същ ред, то придвижването е еднозначно определено и ще бъде само по този ред. Аналогичен е случаят, когато клетките са върху една и съща колонка. Ако клетките са върху различни редове и колонки, тогава следващият ход трябва да се направи така, че всеки нов ход максимално да намалява разстоянието до крайната клетка. Затова трябва да се избира ход, който намалява едновременно разликите в координатите по двете оси, т.е. придвижването ще се извърши по една от четирите диагонални посоки. Като краен вариант стратегията за намиране на път с минимална дължина може да се формулира така: придвижването се извършва по диагонал, който на всяка стъпка максимално скъсява разстоянието до крайната позиция. Това продължава до достигане на позиция, която е на един и същ ред или колонка с крайната позиция, и останалата част от пътя се изминава по тях – съответния ред или колонка.

Съгласно тази стратегия изборът на съседно поле може да се извърши по алгоритъма, който е представен по-долу (Азълов & Златарова, 2011).

Нека *start* и *end* са началната и крайната клетка върху грида, за които се търси път с минимална дължина.

1. Пресмятат се разликите на *x*-координатите на двете позиции: dx = end.x – start.x

2. Пресмятат се разликите на *y*-координатите на двете позиции: dy = end.y – start.y

3. Ако dx = 0, позициите са върху една колонка и придвижването ще се извърши по нея до достигането на крайната позиция.

4. Ако dy = 0, позициите са върху един ред и придвижването ще се извърши по него до достигането на крайната позиция.

5. Ако dx ≠ 0, тогава се пресмята изразът $p = \frac{end.x - start.x}{|end.x - start.x|}$

6. Ако dx ≠ 0, тогава се пресмята изразът $p = \frac{end.y - start.y}{|end.y - start.y|}$

7. Пресмятане на координатите на следващата позиция:

start.x = start.x + p

start.y = start.y + q

Възможните стойности за p са –1, 0 и +1. Ако $p = -1$, тогава съгласно т.7 *x*-координата ще намалее с 1 и придвижването ще е наляво. Ако $p = +1$, тогава съгласно т.7 *x*-координата ще нанарасне с 1 и придвижването ще е надясно. Аналогичен е случаят със стойността на q, която може да бъде –1, 0 и +1. При $q = -1$ придвижването е надолу, а при $q = +1$ ще е нагоре. Като се комбинират всички случаи за p и q, включително и тези, при които те са нули, се получават всичките осем възможни придвижвания.

Дължината на пътя може да е твърде голямо число и използването на динамичен масив (обект от клас вектор) с базов тип *Coord* е добра идея. Това е илюстрирано като пример във функцията *minPath*, която съхранява пътя на фигурата.

```
vector<Coord> minPath(Coord start, Coord end)
{
   vector<Coord> path;              // path – локален вектор за път
                                       (от start до end)
   path.push_back(start);           // Добавяне на началната
                                       позиция към path
   if (start == end) return path;

   int p, q, dx, dy;                // Променливи, използвани за определяне
   do                               // на координатите на следващата клетка
   {
```

```
        dx = end.x - start.x;
        dy = end.y - start.y;
        p = (dx == 0)? 0 : dx/abs(dx);   // p - Посока за придвижване по x
        q = (dy == 0)? 0 : dy/abs(dy);   // q - Посока за придвижване по y
        start.x = start.x + p;           // start.x – нова стойност на x-координата
        start.y = start.y + q;           // start.y – нова стойност на y-координата
        path.push_back(start);           // Добавяне на нова позиция към пътя
    } while (start == end));             // Критерий за достигане на крайната позиция
    return path;
}
```

Фигура 13. Намиране на най-кратък път между две клетки от грид

Не е трудно да се съобрази, че пътят с минимална дължина между две произволни клетки $A(a_1, a_2)$ и $B(b_1, b_2)$ от грид се пресмята чрез израза (разстояние на Чебишев):

$max\{|a_1 - b_1|, |a_2 - b_2|\}$,

който по дефиниция ще приемаме да бъде разстоянието $dist(A, B)$ между две произволни клетки A и B от грида.

Играта „Преследване“

В играта G2.3 участват двама играчи, чиито фигури A и B имат случайно избрани начални позиции върху грида. Целта на фигура A е да „хване“ фигурата B, като „стъпи“ на нейната текуща клетка. Но фигурата B „вижда“ A и иска да „избяга“ от нея. Затова след всеки ход на A тя се старае, ако е възможно, да се премести в съседна клетка, която е на по-голямо (не по-малко от предишното) разстояние до A. Съседни на една клетка са клетките, с които тя има обща страна или връх.

Стратегии на участващите в играта. Фигурата A иска да скъси разстоянието между себе си и фигура B до минимум, т.е. до 0. В този случай фигура A ще е „хванала“ фигура B. Обратно, фигура B се стреми с всеки ход да се премести в клетка, която я отдалечава от A, ако това е възможно. И в двете стратегии става дума за разстояние между клетки от грид. Чрез функцията „разстояние между клетки на грид“ *dist* двете стратеги за ход могат да се запишат по следния начин:

$dist(A', B) < dist(A, B)$ – стратегия за ход за преместване на A в A'

$dist(A, B') \geq dist(A, B)$ – стратегия за ход за преместване на B в B'

На базата на тези стратегии са написани функциите, с които се избира ход на двамата играчи. Първият играч, да го именуваме *Tom*, преследва вторият играч – *Jerry*.

Избор на ход на Том

```
void GameAlg::game2P0nextMove (Game& game)
{
  Coord a = game.getPlayers()[0].getCurrPos();
  Coord b = game.getPlayers()[1].getCurrPos();

  int dx, dy;
  dx = b.x - a.x;
  dy = b.y - a.y;
  a.x = a.x + ((dx == 0)? 0 : dx/abs(dx));
  a.y = a.y + ((dy == 0)? 0 : dy/abs(dy));
  game.getPlayers()[0].setCurrPos(a);
}
```

Фигура 14. Функция, дефинираща стратегията на Tom

Избор на ход на Jerry

Множеството *xy* на всички клетки, съседни на дадена клетка с координати (*x*, *y*), е:

$$xy = \{x + NEXT8[i][0], y + NEXT8[i][1]\}$$

Масивът NEXT8 е дефиниран по следния начин:

```
const int N8 = 8;
const int NEXT8[N8][2] = {{ 1,0}, { 1,-1}, {0,-1}, {-1,-1},
                          {-1,0}, {-1, 1}, {0, 1}, { 1, 1}};
```

Някои клетки на множеството *xy* може да са извън грида и това се проверява с функцията *inSide*. *Jerry* трябва да избере клетка, която е по-отдалечена от клетката, в която е *Tom*. Такава клетка може и да не съществува, а може и да не е единствена.

```
void GameAlg::game2P1nextMove(Game& game)
{
  Coord c;
  Coord a = game.getPlayers()[0].getCurrPos();
  Coord b = game.getPlayers()[1].getCurrPos();
  Grid grid = game.getGrid();

  int d;
  int inx = 0;
  double max_dist = -1;
```

```
  for (int i=0; i<N8; i++)
  {
    c.x = b.x + NEXT8[i][0];
    c.y = b.y + NEXT8[i][1];
    if (!grid.inSide(c)) continue;

    d = max(abs(c.x - a.x), abs(c.y - a.y));
    if (max_dist < d)
    {
      inx = i;
      max_dist = d;
    }
  }
  c.x = b.x + NEXT8[inx][0];
  c.y = b.y + NEXT8[inx][1];

  game.getPlayers()[1].setCurrPos(c);
}
```

Фигура 15. Функция, дефинираща стратегията на Jerry

Краят на партия от играта „Преследване“ настъпва, когато позициите на играчите съвпаднат (фиг. 16):

```
bool GameAlg::game2End (Coord a, Coord b) const
{
  return a == b;
}
```

Фигура 16. Функция за проверка за край на играта

7. Проект в развитие

Да разгледаме развитието на партията от фиг. 17. Вижда се, че и двамата играчи играят по правилата, заложени в техните стратегии – функцията *game2P0nextMove* за *Tom* и функцията *game2P1nextMove* за *Jerry*. *Tom* „хваща“ *Jerry* още на петия ход. Това развитие на играта поражда интересни въпроси, изследването на които може съществено да подобри стратегията и на двамата играчи. Ето няколко конкретни насоки за изследване, които са формулирани като задачи.

Задача 1. [*Край на играта*] Винаги ли *Tom* „хваща“ *Jerry*? Ако отговорът е „Да“, то какъв е минималният брой ходове за това? Ако отговорът е „Не“, да се посочи начална позиция и стратегия на *Jerry*, при която той е недостижим.

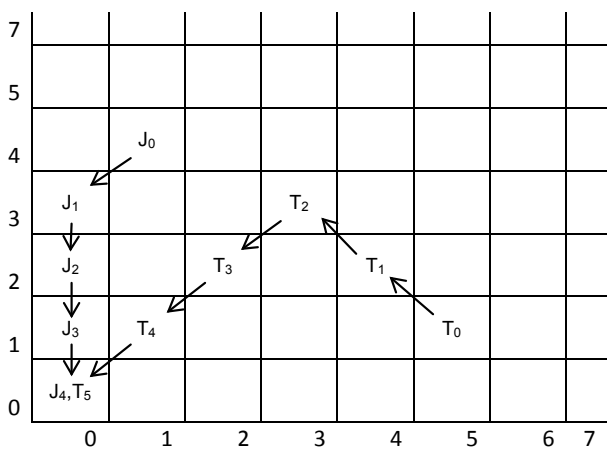
Задача 2. [*Подобрена стратегия на Jerry*] Изборът на ход на *Jerry* невинаги е еднозначно определен. Може ли измежду всичките възможни ходове, които го отдалечават от текущата позиция на *Tom*, да се избере такъв, който му гарантира повече ходове, преди да бъде хванат?

Задача 3. [*Jerry с „изкуствен интелект“*] Да се формулира нова стратегия на *Jerry*, която ще „имплантира“ интелект, с който да се вземат по-добри решения във всеки момент от развитието на партията.

Задача 4. [*Подобрена стратегия на Tom*] Изборът на ход на *Tom* не е винаги еднозначно определен. Може ли измежду всички възможни ходове, които скъсяват пътя му до текущата позиция на *Jerry*, да се избере ход, който му гарантира по-бързото „хващане“ на *Jerry*?

Задача 5. [*Tom с „изкуствен интелект“*] Да се формулира нова стратегия за *Tom*, която ще му „имплантира“ интелект, с който да взема по-добри решения във всеки момент от развитието на партията.

Задача 6. [*Игри върху грид с недостъпни полета*] Игрите G2.1-G2.5 могат да се формулират, реализират и играят в много по-сложен вариант, при който в грида има недостъпни полета. Някои от тези игри са модели на реално съществуващи компютърни игри. Основна-



Фигула 17. Илюстрация на партия от играта „Преследване“

```
The game starts

Number of players = 2
Grid's dimensions = 15 x 5
Player name: Tom
Initial position: [5, 1]
Player name: Jerry
Initial position: [1, 4]

Move 1
Tom:   [4, 2]
Jerry:  [0, 3]

Move 2
Tom:   [3, 3]
Jerry:  [0, 2]

Move 3
Tom:   [2, 2]
Jerry:  [0, 1]

Move 4
Tom:   [1, 1]
Jerry:  [1, 0]

Move 5
Tom:   [1, 0]

Tom is a great cat
```

та трудност при тях са алгоритмите, с които се извършва ход на дадена фигура върху грида. Тези алгоритми не са тривиални и най-често са алгоритми за търсене в граф. Наред с класическите алгоритми за търсене в ширина и дълбочина, някои игри използват техни модификации, в които има вградени евристики, като например *алгоритъма A** [4].

В класа *Grid* има функция (*setInaccCells*), с която по случаен начин се генерира грид с *INACCESSIBLE* (цяло положително число) процента недостъпни полета. В програмния текст от фиг. 9 като коментар е даден фрагмент, който трябва да се вмъкне при инициализирането на игра, в която гридът е с недостъпни клетки. Глобалната величина *INACCESSIBLE* е параметър в игрите с грид и е декларирана в отделен файл.

Вместо заключение

Тръгвайки от конкретна математическа задача, описана като игра в (Чуканов, 1975), в статията е представен модел за изграждане на компютърни игри върху грид. Известно е, че разработването на професионални компютърни игри изисква решаването на множество трудни задачи. Част от тях обикновено са чисто математически. Но тъй като невинаги е възможно да се намери и математически да се докаже наличието на печеливша стратегия за много игри, най-често вземането на решение за ход се основава на изследването на възможните ходове и на двамата играчи. В общия случай броят на тези ходове е огромен и това налага да се разработват евристични алгоритми, които да генерират приемливи решения за разумно кратко време.

БЕЛЕЖКИ

1. Играта има математически анализ, съгласно който първият играч никога не губи, т.е. в най-лошия вариант за него играта завършва наравно (Чуканов, 1975).
2. Amit's Game Programming Information. http://www-cs-students.stanford.edu/~amitp/gameprog.html

ЛИТЕРАТУРА

Азълов, П. & Златарова, Ф. (2011). *C++ в примери, задачи и приложения*. София: „Просвета".

Чуканов, В. (1975). *Математически игри.* София: „Народна просвета".

Grozdev, S. & Doichev, S. (2005). A didactical system of entertainment problems for the development of skills, Proc. Int. Conf. on Mathematics Education, Svishtov, June 3 – 5 2005, pp. 114 – 119.

Leuteneger, S. & Edgington, J. (2007). A Game First Approach to Teaching introductory Programming, *Proc. ACM, SIGCSE'07*, pp.115 – 118.

REFERENCES

Azalov, P. & Zlatarova, F. (2011). *C++ v primeri, zadachi i prilozheniya.* Sofiya: Prosveta.

Chukanov, V. (1975). *Matematicheski igri.* Sofiya: "Narodna prosveta".

Grozdev, S. & Doichev, S. (2005). A didactical system of entertainment problems for the development of skills, Proc. Int. Conf. on Mathematics Education, Svishtov, June 3 – 5 2005, pp. 114 – 119.

Leuteneger, S. & Edgington, J. (2007). A Game First Approach to Teaching introductory Programming, *Proc. ACM, SIGCSE'07*, pp.115 – 118.

FROM MATHEMATICAL TO COMPUTER GAMES

Abstract. Educational games are used not only for acquisition of new factual knowledge but also for developing logical thinking and decision making in mostly uncommon virtual environment. The creation of such games has also its own educational merits and this aspect is the key accent of this paper.

Leading from a specific mathematical problem, redefined as a game, the paper lays out a model for building a set of computer games on grid. This model is presented by developing a set of program classes. They are used for designing the grid and the players of a game.

The examples considered in this paper are based on two specific games that use the program classes described above. Each game has a specific grid initialization and a corresponding configuration of the participating players/figures. The rules of how players „move on the grid" should also be determined in advance together with the criteria for determining when of the end of the game is reached and who the winner is, if any.

A set of sample computer games illustrates the ideas described in the text. Using the model in consideration as a foundation, the paper shares ideas for developing future computer games on grid. Some of these games have possible research value and could be explored as interesting and in some cases challenging projects.

✉ **Dr. Pavel Azalov, Assoc. Prof.**
Pennsylvania State University
Hazleton campus, U.S.A.