|  |  |
| --- | --- |
| Картина, която съдържа текст, вентилатор, устройство  Описанието е генерирано автоматично | **ВИСШЕ ВОЕННОМОРСКО УЧИЛИЩЕ „Н. Й. ВАПЦАРОВ“**  ***9002 Варна, ул. „В. Друмев“ 73, тел.052/632-015, факс 052/303-163*** |
| ***“FILII MARIS SUMUS”***  **ФАКУЛТЕТ „ИНЖЕНЕРЕН“**  **КАТЕДРА „ИНФОРМАЦИОННИ ТЕХНОЛОГИИ или ЕЛЕКТРОНИКА“** |
| A picture containing text  Description automatically generated |  |

**KУРСОВ ПРОЕКТ**

**по дисциплината „Основи на програмирането и алгоритмизация“**

**теМА: „CLASSIC ARCADE GAME CLONE: RECREATING A CLASSIC ARCADE GAME LIKE PAC-MAN OR TETRIS“**

**Студент: Дамян Веселинов Чакъров**

**Специалност: Софтуерни и Мрежови технологии**

**фак. № 141-24115**

**гр. Варна**

**2024 г.**

Този проект е терминално пресъздаване на аркадната игра Тетрис на езика C. Тетрис представлява поле, което се изпълва с фигури, подредени от играча, които са винаги с абсолютен размер 4 блокчета и се различават единствено по подредбата и цвета им.

**Техническа документация**

Програмата е съставена от 3 файла:

* main.c
* functions.c – изпълнението на всяка една от функциите в програмата, като двете основни, към които се директно се достъпва от main файла, са play() и scoreboard().
* globals.c – декларирането на всяка глобалните променливи за програмата се случва тук, както и дефинирането на константи.

Допълнителни 2 header файла:

* functions.h – инициализацията на функциите, описани в едноименния .c файл.
* globals.h – служи като връзка между globals.c и останалите файлове в програмата, за да може да се достъпва в променливите.

В програмата освен двата създадени от мен header файлове са използвани и допълнително 5, от които:

<stdio.h>

Използвани са стандартните функции за запис и четене от конзолата/файла.

<stdlib.h>

Използвани са scanf(), system(), rand() и srand().

<stdbool.h>

Използвани са променливите от типа bool.

<string.h>

Използвана е функцията memcpy().

<unistd.h>

Използвана е функцията sleep().

<time.h>

Използвана е time().

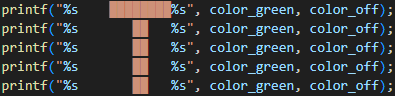
<conio.h>

Използвани са getch() и kbhit()

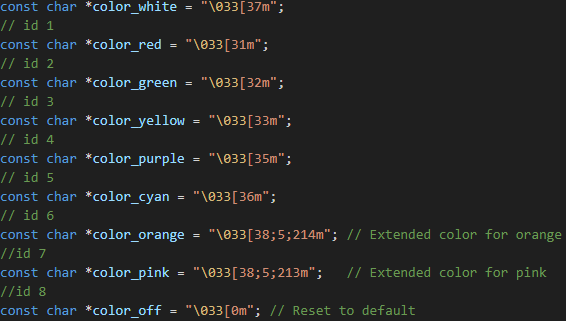
**Изпълнението на main() в main.c файла:**

Започва с декларирането на char input;, в която ще се съхранява входа от потребителя. След което е използвана функцията system("cls");, за да се изчистят предходни команди и входове в терминала.

Редовете 22-26 изписват заглавното име на играта TETRIS в различни цветове, за пример с Т:

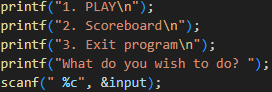


символът на издължен правоъгълник, който е използван е ограден от две %s, чрез които се вмъкват константните променливи от globals.c (с връзка чрез globals.h), съотвестващи на избрания цвят (или липса на такъв и отново върнати към стандарта).

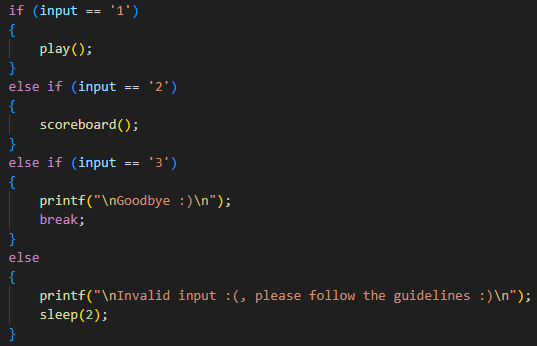


Декларацията на константите за цвета следва ANSI принципа, че първата част от декларацията при всеки един от цветовете \033[ инструктира терминала за предстояща промяна на цвета, стила или позицията, а следходното е желаната промяна, в случая са кодовете за всеки от съответните цветове.

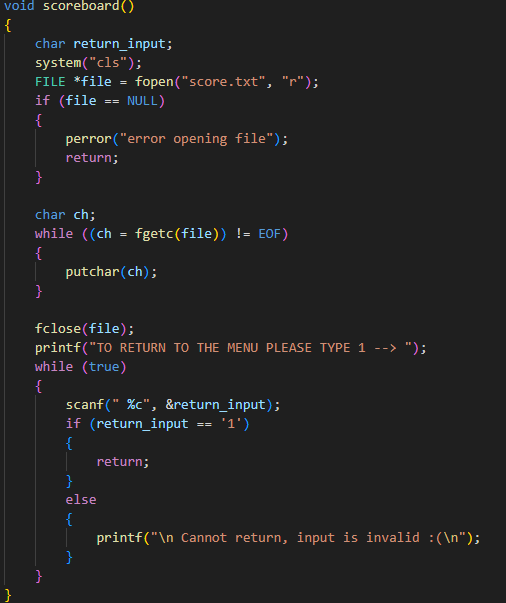
След изписването на заглавното лого на приложението следват

опциите, които са представени на потребителя, като след това се разчита избраното и се записва в адреса на променливата. Функцията разчита всичко въведено от потребителя, включително и новия ред Enter, за да се избегнат нежелани входове в променливата е поставено пространство преди %c, което сигнализира на функцията да пропусне whitespace characters.

Проверката на входа се изпълнява по следния начин:



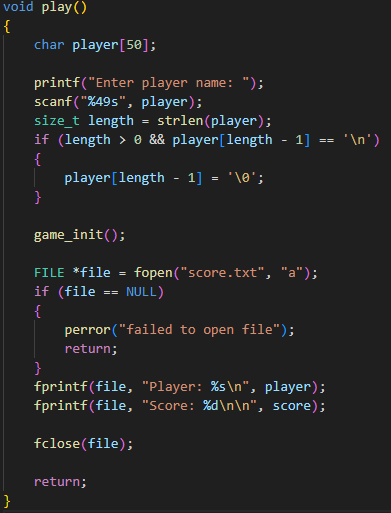
При невалиден вход, програмата извежда съобщение, изчаква 2 секудни и след това отново се изчаква за вход от потребителя.

При начален вход ‘3‘, тоест изход, се излиза ефективно от програмата и се завършва успешно изпълнението.

При начален вход ‘2‘, се извиква функцията scoreboard() от functions.h като същинския код на функцията се съдържа в functions.c.

В нея - декларира се променлива char return\_input;, след което се изчиства терминала и се отваря файла score.txt за прочит. Прави се проверка дали е успешно отварянето на файла, ако не – return;, а в противен случай се изписва цялостния файл чрез обхождане. Изисква се вход ‘1‘ за да се върне програмата обратно в менюто.

При начален вход ‘1‘ се извиква функцията play(), от която следва същинската част на проекта.



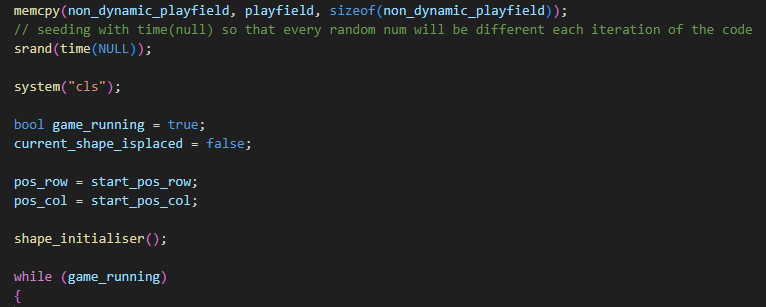
Тя започва с деклариране на масив от типа char, където ще се съхрани името на потребителя, като чрез if() проверката се премахва празното пространство в края.

След успешното съхранение на името се извиква функцията game\_init(), която е входната точка на играта.

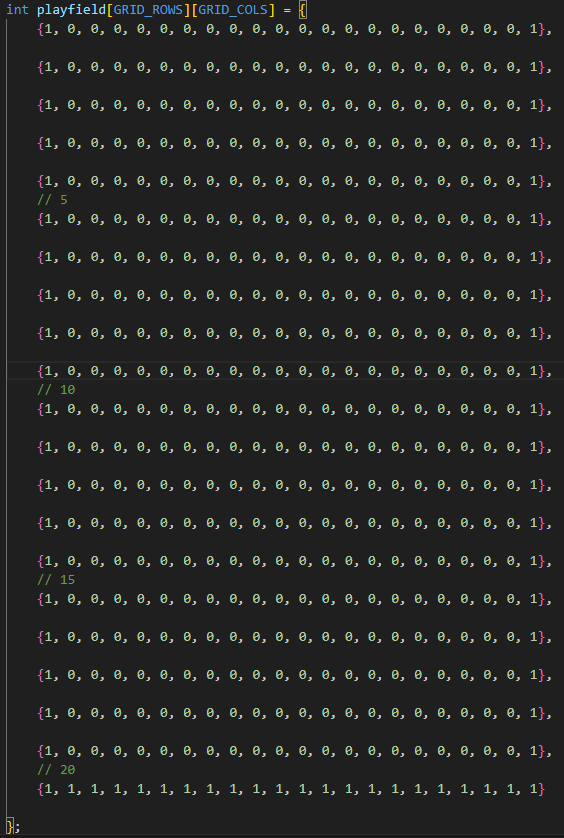
След прекратяването на играта, се отваря файлът score.txt, но тук със свойството append (да се изписва текст на края на файла). Прави се проверка дали успешно е отворен файлът – ако да - се изписвсат името на играча и неговия резултат, използвайки глобалната променлива score , след което файлът се затваря и потребителят е отново върнат в менюто на програмата.

game\_init()

Това е функцията, в която върви самият цикъл на играта.



Преди влизането в while(), се прави подготовката за първата инстанция в играта :



Копират се стойностите на двумерния масив playfield[][] в преди това празния non\_dynamic\_playfield[GRID\_ROWS][GRID\_COLS];, където 0 обозначава празните позиции, а с 1 границите на полето за игра. Константите по които са дефинирани двата двумерни масива

#define GRID\_ROWS 21

#define GRID\_COLS 22

го определят по такъв начин, че ако се премахнат двете колони с единици за самата клетка и последния ред, би се образувал квадрат от нули. Тъй като символът който се използва за изобразяването на всяка една от фигурите в полето :

█

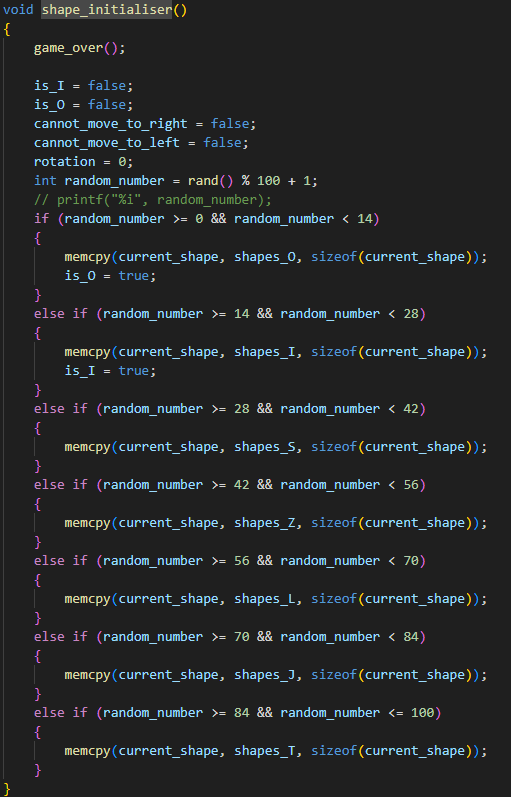
е по-висока отколкото широка, за да се постигне желаната формана кубче, поставяме два пъти този символ

██

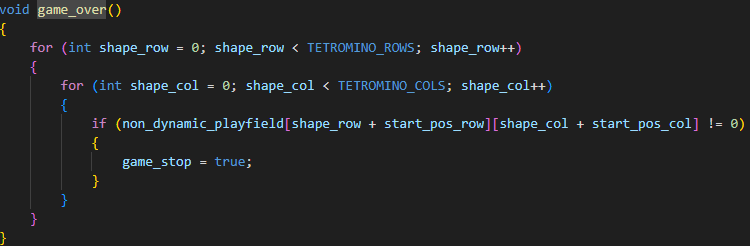
, което е и причината полето, което на пръв поглед е правоъгълно, реално да е квадрат.

Булевата променлива, която контролира дали играта ще върви се инициализира и задава на true, освен това задаваме false на проверката дали текущата фигура е поставена.

Координатите на които ще се призове тази именно фигура се задават на стартовите (декларирани в globals.c и са с позиция нулев ред, девета колона) и се извиква функцията shape\_initialiser(); :



Първата стъпка при опита да се генерира нова фигура, е да се провери дали при опит да се призове, няма да се припокрие с вече поставена предходна фигура:



Чрез проверката се обхождат координатите, където ще се позиционира следващата фигура, която е с размери:

#define TETROMINO\_ROWS 4

#define TETROMINO\_COLS 8

По този начин проверяваме правоъгълник 4х8 спрямо началната точка и ако на някоя от съответните позиции има стойност, различна от 0 (друга фигура), булевата променлива за прекъсването на играта се задава на true.

Ако проверката е преминала успешно, тоест не се припокриват фигурите, променливите :

is\_I, is\_O, cannot\_move\_to\_right, cannot\_move\_to\_left

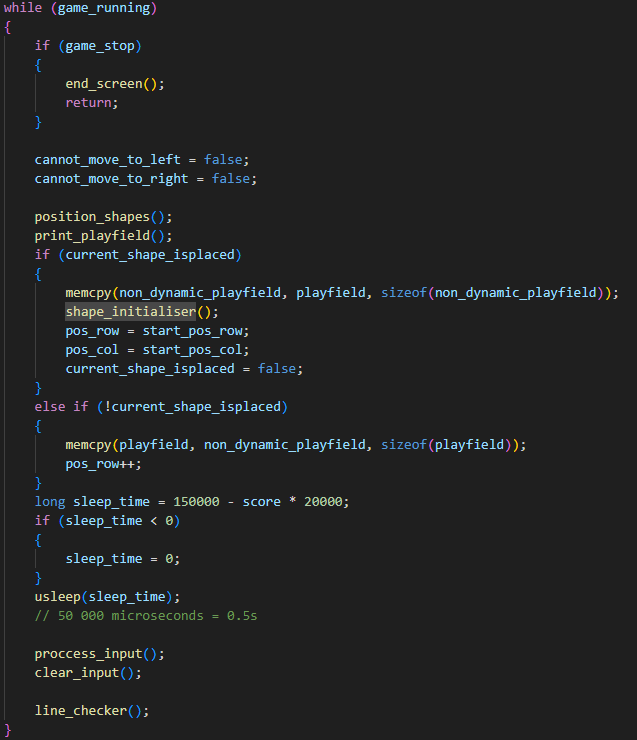
отново се задават на false, за да няма непредсказуемо поведение при по-нататъчна употреба, а rotation = 0; за да се призове в стартовата си ротация съответната фигура.

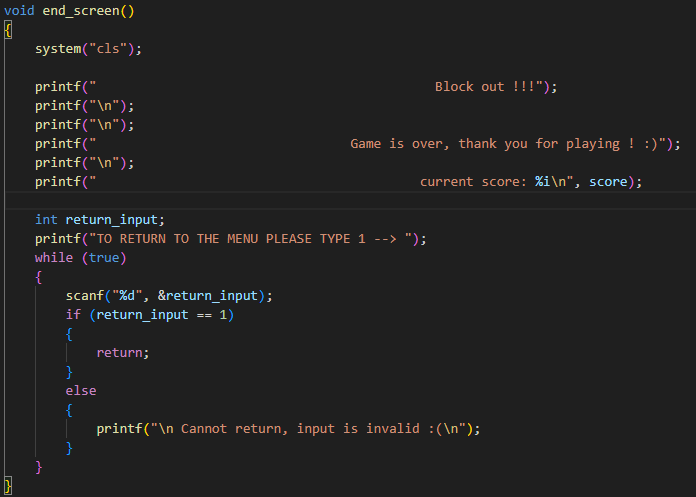
Следващата стъпка е да се генерира произволно число, спрямо което да се определи типа на следващата фигура.

След като се определи каква ще бъде, се копират и 4те форми на избраната фигура в тримерния масив current\_shape.

За фигурите квадрат и I, са направени булеви променливи заради, по-специфични ситуации при завъртането на двете фигури.

При вече извършена заготовка, се влиза в while цикъла на играта:



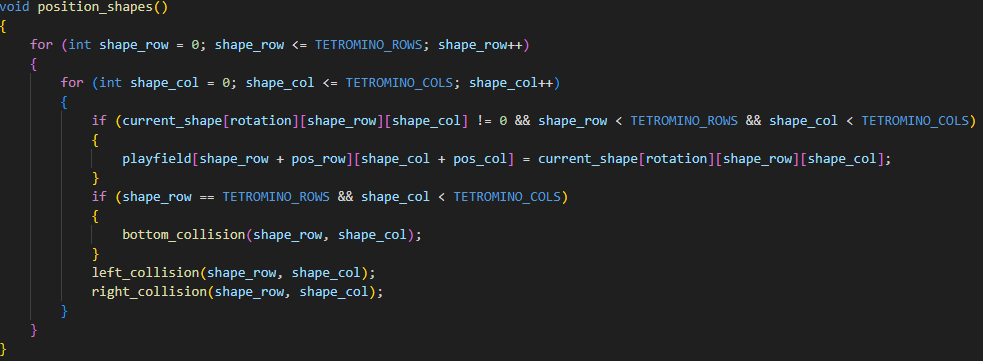


При всеки един цикъл на играта първо се прави проверка дали булевата променлива game\_stop не е true. Тя би била със такава стойност само ако при опит на shape\_initialiser да генерира нова фигура, не би се припокрило с вече поставена такава на полето. Ако, обаче, да приемем че сме на по-напреднал етап от играта и game\_stop е true, се извиква функцията end\_screen(); , която извежда колко точки сме постигнали и ни връща отново в заглавното меню.

След проверката, булевите променливи за движение в двете посоки

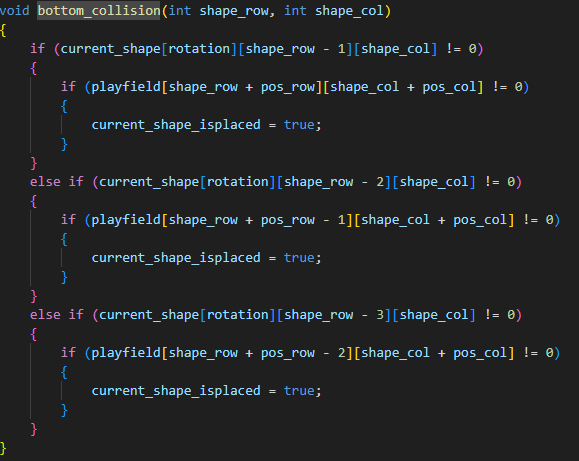
cannot\_move\_to\_left = false;

cannot\_move\_to\_right = false;

 отново се поставят на false, тоест МОЖЕ да се отива и на двете посоки. А след това се извиква функцията position\_shapes()

Тук се задават стойностите на текущата фигура спрямо началните координати в полето playfield. Както си спомняме, преди да влезем в while цикъла, ние уеднаквихме двата масива за полето playfield и non\_dynamic\_playfield, с целта, че в процеса на работа на играта, non\_dynamic\_playfield ще съдържа САМО вече поставените фигури на полето, а playfield – същото като non\_dynamic\_playfield + текущата фигура, която потребителя все още може да контролира (докато не се постави и тя и играча ще приеме контрол над най-скорошната фигура).

При всяко позициониране на фигурите, тоест задаване на стойностите на текущата фигура в динамичния playfield, се прави проверка за колизиите на текущата фигура чрез bottom\_collision(), left\_collision() и right\_collision().

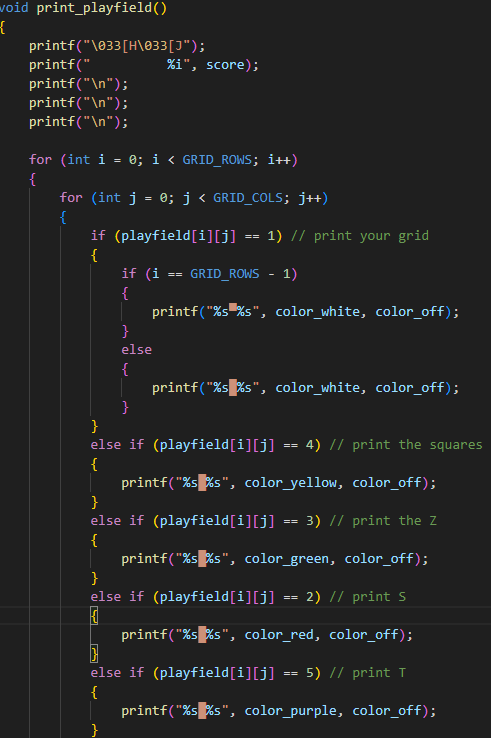


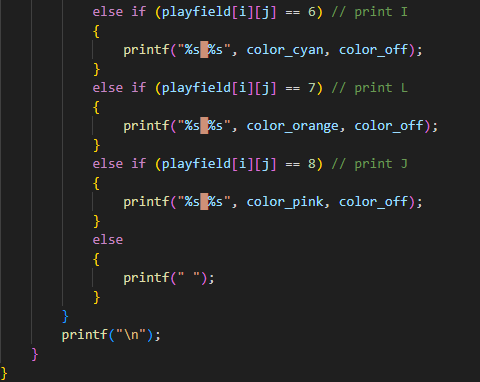
Чрез bottom\_collision() се променя

current\_shape\_isplaced спрямо това дали точно под най-долния елемент на текущата фигура има друга, вече поставена фигура. Ако да – се променя булевата променлива на true.

Регулирането на това, дали фигурата ще може да излиза от заграждението на полето от единици се определя от left\_collision() и right\_collision() които, поставят съответстващите булеви променливи на true, ако това би попречило на цялостта на полето.

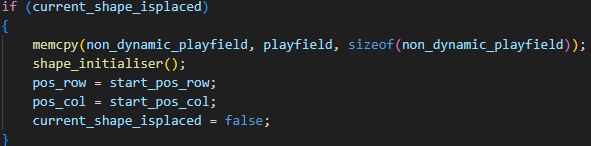
Следващата стъпка е извикването на функцията print\_playfield() :



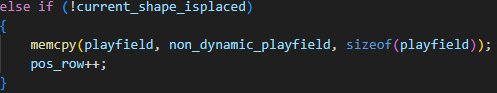


В тази функция се принтира полето, включително с текущата контролируема от играча фигура, в зависимост с желания цвят спрямо цифрата индекс. Ако стойността е различна от избраните, тоест 0, се изобразява „ “ – празно пространство.

Следващата проверка е за това, дали текущата фигура би трябвало да бъде поставена. Тази булева променлива вече е променена в bottom\_collision() и единствено следва да бъде се копира състоянието на динамичното поле playfield съответно в non\_dynamic\_playfield – с цел да се запази състоянието на играта, и да се върнат координатите на следващата фигура на началните, както и да се върне проверката за поставянето на фигурата отново на false.



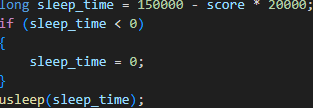
Ако фигурата не е поставена, обаче :



се копира състоянието на non\_dynamic\_playfield на мястото на динамичния playfield. Целта на тази част от кода е:

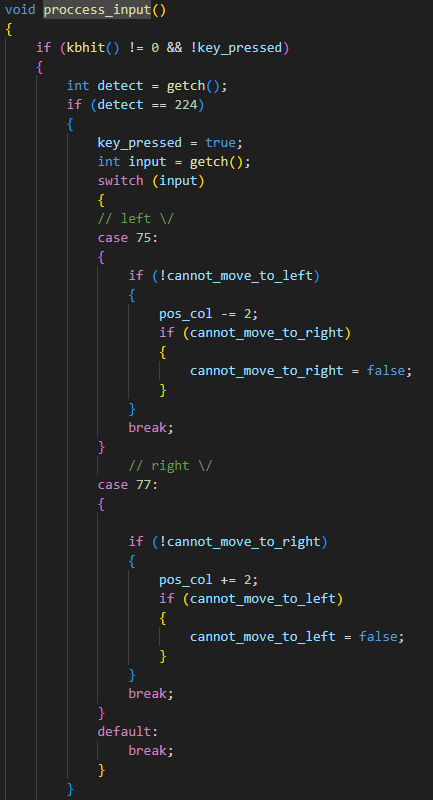
* тъй като контролируемата от играча фигура се мести с една позиция надолу с всеки цикъл на играта, да се премахне предходното нейно състояние, тъй като то не е вече нужно. Тоест – програмата зависи главно от non\_dynamic\_playfield за да определя коя фигура къде е, а динамичния playfield се изобразява единствено за да може потребителя да се ориентира къде се намира текущата фигура и за да може да я премести на желаната позиция.
* да измести позицията на фигурата с един ред надолу.

Следва



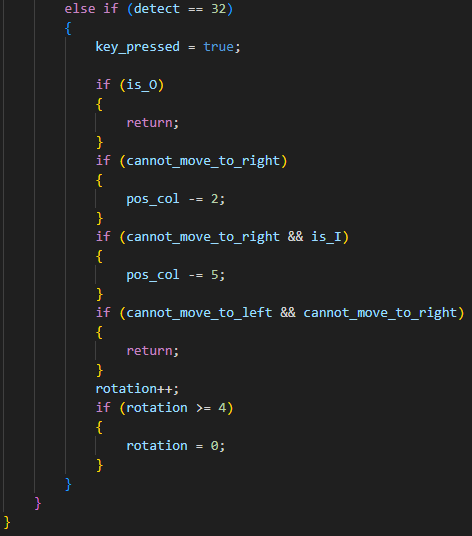
Тук се определя с каква скорост ще се опреснява екрана. В началото скоростта е 150хиляди микросекунди, тоест всеки 1.5 секунди ще се опресни екрана, а с напредването на играта, се ускорява с 0.2 секунди на всяка направена точка резултат, до минимална граница 0 секунди. Изчакването между всяко опресняване се прави от функцията usleep.

След изчакването следва функцията process\_input(), спрямо която, ако потребителят е използвал някой от клавишите за контролиране на фигурата – space, left arrow или right arrow се променя състоянието на фигурата подобаващо:

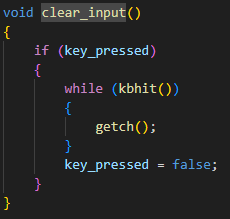


Кодът за стрелките на клавиатурата е 224 ХХ, тоест се подават две стойности на променливата detect и именно затова са напоравени две проверки, ако стойността е 224, се отчитат като вход от стрелките и се използва getch() още веднъж с цел да се вземе и следващата стойност. За движение на ляво е при кодът 75, и ако спрямо булевата променлива МОЖЕ да се придвижи наляво, се придвижва, а ако спрямо булевата променлива за придвижване надясно НЕ МОЖЕ да се придвижи надясно в текущото състояние, тя се поставя на false. Аналогично при движението за надясно.

Тоест – при position\_shapes функцията булевите променливи за възможността да се придвижи ако са били променени, в тази функция не бихме могли да изпълним придвижване, а ако изпълним придвижване на фигурата, трябва да се провери булевата променлива за придвижване в обратната посока, защото ако досега сме били прилепени до примерно дясната стена, не можем да отидем по-надясно, НО ако се придвижим и веднъж наляво, вече отново има място накъдето да се придвижим надясно.

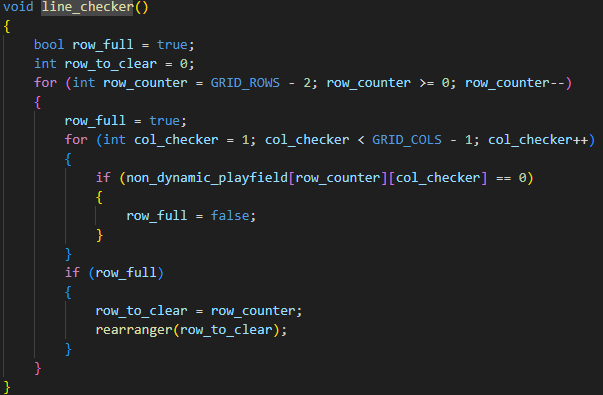
 Ако входът от потребителя не се отнася за стрелките се прави проверка за кода на spacebar, който е отговорен за завъртането на фигурата. Допълнителните проверки се изпълняват за да се предотврати излизането на фигурите от очертянията или припокриването им с други вече поставени фигури при завъртането. Ако са преминати успешно проверките, се добавя единица към индекса на завъртане, тоест се преминава на следващото завъртане.

След успешно или неуспешно – без значение – разчитане на вход от потребителя се изчистват променливите за отчитане на вход:



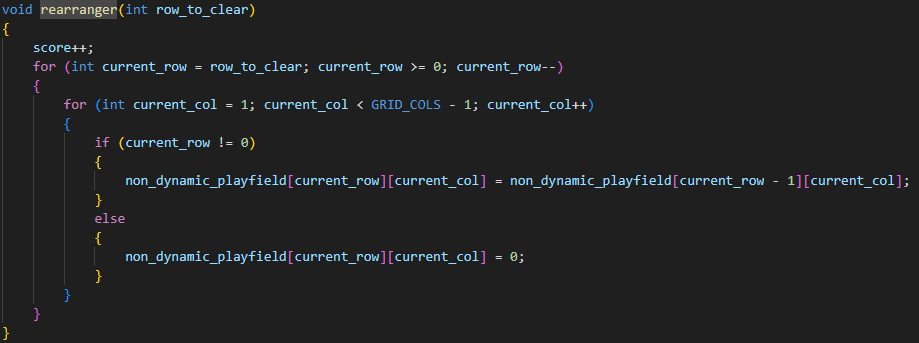
Булевата променлива съществува и се използва в предходната функция process\_input() за да се избегнат множество промени на състоянието в един цикъл на програмата.

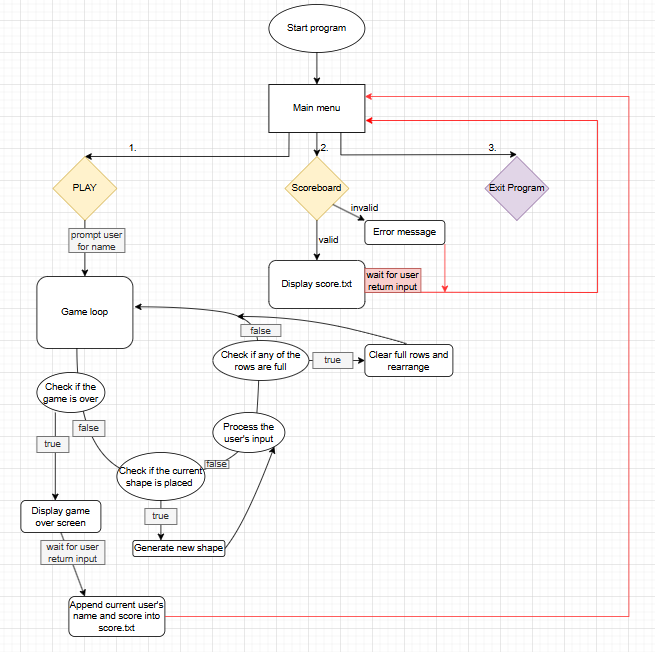
Последната функция, която се извиква в цикъла на играта е:



Тук се задава булева променлива за проверка на всеки ред, като се започва от най-долния, преминавайки с един ред нагоре при всяко завъртане, и се проверяват елементите във всяка една от колоните му. Ако в който и да е елемент от този ред има празно пространство 0, се задава на false.

Ако, обаче, няма празно пространство 0 се маркира текущия ред като нужен да се изчисти на променливата row\_to\_clear и се извиква функцията rearranger().

 При всеки ред, който е изчистен в хода на програмата се прибавя 1 към крайния резултат на потребителя. Самото изчистване на реда се случва като на текущия маркиран за изчистване ред се задават стойностите на реда, който се намира директно над него, и така, докато не стигнем до най-горния ред, където се задават 0 на всеки от неговите елементи.

**UML Диаграма:**

**Бъдещи подобрения:**

* Възможност да се направи преглед за следващо блокче, което ще бъде призовано.
* Поредност и редуване на фигурите, за да се избегне възможността да се генерират в последователност еднакви фигури.
* Подобрен начин на изобразяване на самата игра, за да се избегне постоянното пробляскване на екрана.
* Подобряване на разчитането за движение на текущата фигура от потребителя. Възможност да се изпълняват множество движения наляво или дясно преди да се придвижи фигурата надолу.
* Възможност да се поставя фигурата автоматично на най-долния ред.